ICOT Technical Report: TR-602

TR-602

制約とマルチコンテクストに基づく並列協調問題解決

横山　孝典，小野　昌之，和田　正寛，
大崎　宏 (JIPDEC)

November, 1990

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

制約とマルチコンテクストに基づく並列協調問題解決

横山　孝典　　小野 昌之　　和田　正寛
（財）新世代コンピュータ技術開発機構
大崎 宏
（財）日本情報処理開発協会

梗概

本論文では、部分問題間の依存性が大きな設計問題向きの並列協調問題解決方式について述べる。我々は部分問題間の依存関係は設計対象の共有により生ずることに着目し、設計対象のもつ制約を利用した協調問題解決デルを提案する。このモデルでは制約充足機能を有するオブジェクトで表現した設計対象モデルを中心に、複数のエージェントが並列かつ協調して問題解決を進める。そして制約充足機能により、設計解の整合性を保つとともに、制約伝播を利用した協調動作を実現可能としている。また、スロットの組み合わせを扱うノード網を用いた効率的なマルチコンテクスト機能をオブジェクトに持たせることにより、複数の解候補を並列に処理し、処理速度の向上を図っている。そして、この方式に基づいた並列協調設計システムのプロトタイプを並列論理型言語ＫＬ１を用いて、分散メモリ型のマルチプロセッサ計算機Multi-PSI上に試作し、その有効性を確認した。

1. はじめに

知識処理の重要な応用分野として設計がある。設計は一定の制約条件のもとで、要求仕様を満足する解を生成する問題である。しかし、現実の設計、特に概念設計、方式設計、機能設計等の上流過程は複雑で、問題全体をそのままの形で扱うことは困難である。

そこで、定式化が可能な部分問題に分割して部分解を求め、それらを合成して全体の解を得るという手法が用いられる[1]。しかし、部分問題間に存在する依存関係のため、それらを完全に独立に解くことはできないのが普通である。また、合成した解の整合性のチェックが必要なことや、問題によっては部分解がトレードオフの関係にあることがあり、単純に部分解を合成して全体の解を得ることはできない。

このため、複数の知識源あるいはエージェントが部分問題を処理し、協力して設計解を生成するという、協調問題解決が有力となる[2]、[3]、[4]。代表的な協調問題解決方式に黒板モデルがあり[5]、[6]）、黒板モデルに基づいた協調設計システムがいくつか発表されている[7]、[8]、[9]。黒板モデルは概念的には複数の知識源が並列に推論を進めることができ、並列化による実行速度の向上も期待される。

しかし従来の黒板は基本的には静的な共有メモリであり、協調動作を実現するには、次にどの知識源の処理を実行するかを決定するスケジューリング機構や、合成した解の整合性をチェックするための知識源を別に設ける必要がある。しかし、効率よいスケジューリングの実現は簡単でないうえ、スケジューリングの知識の獲得及び管理が難しいという問題がある。また、スケジューリングの導入により処理が逐次的になりやすい、黒板へのアクセスがボトルネックになりやすいなど、並列化における問題も多い。

我々は、部分問題間の依存関係は設計対象の共有により生ずることに着目し、設計対象のもつ制約を利用した協調問題解決モデルを提案し、それに基づく設計システムを開発中である。このモデルでは制約充

足機能を有する設計対象モデルを中心に、複数のエージェントが並列に問題解決を進める。そして制約充足機能により、設計解の整合性を保つとともに、制約伝播を利用した協調動作を実現可能としている。また、複数の解候補を同時に扱うことのできるマルチコンテクスト・オブジェクトを導入し、効率的な並列処理の実現を図っている。

本論文では上記システムの実現に必要な設計対象表現と問題解決の枠組を提案する。以下ではまず、設計問題における並列協調問題解決の課題について考察する。次に、その課題を実現するため、制約充足機能とマルチコンテクスト機能を持つオブジェクトで表現した設計対象モデルを中心に、複数のエージェントが並列かつ協調的に動作する問題解決モデルを提案する。

そして制約充足に基づく整合性維持と協調動作について述べた後、複数の解候補を並列に処理するためのマルチコンテクスト管理方式について説明する。最後に本方式を用いた設計システムのプロトタイプを紹介する。

2. 設計問題と並列協調問題解決

2.1 設計における協調問題解決

近年、定型的な設計作業のみでなく、試行錯誤的に解を求めなければならない問題に対しても知識処理を用いた自動化が試みられている。そして、複雑な問題に対しては、いくつかの部分問題に分割して部分解を生成し、それらを合成して全体の解を得るという手法が用いられる[1]。

問題の分割のしかたには対象の全体・部分関係に従って部分に分割するという方法と、ひとつの対象を複数の異なる側面に分割するという方法がある。前者はひとつの問題を複数の同質の部分問題に分割するもので、基本的に問題の大きさを縮小することにより問題解決の効率化を図る手法である。例えばマイクロプロッセサをＡＬＵやレジスタなどの部分毎に設計した後、合成する場合がこれに当たる。

一方後者は設計対象を複数の異なる側面で眺め、性質の異なる部分問題に分割するものである。例えばひとつのマイクロプロセッサを動作設計、構造設計、タイミング設計などの観点から設計する場合である。この分割の特徴は、ひとつの設計対象を複数の部分問題で共有する必要があるため、部分問題間の依存性が大きいことである。

この場合には、部分問題を解いた後に解の合成を試みる方式では非効率である。というのは、単純に部分解を合成して整合性のある解を得ることが困難なため、一般に、解を得るまでに何度も後戻りをして部分問題を解き直す必要があるからである。従って、整合をとりながら各部分問題を並行して段階的に設計を進める、協調問題解決が有力になる。ここで、設計における協調問題解決とは、部分問題間で中間解などの情報を互いに交換しながら、並行に設計を進める方式である[2]。

なお、本論文では部分問題の解を部分解、設計途上で生成される部分的な解を中間解と呼ぶことにする。

また、協調問題解決は概念的には複数の部分問題を並列に解くことができる。現実の設計の多くは膨大な計算量を必要としており、並列処理の導入により、効率よい協調問題解決の実現が期待されている。

2.2 協調問題解決モデル

人工知能には分散ＡＩあるいは分散協調問題解決と呼ばれる研究分野があり[3)、4)]、黒板モデル[5)、6)]、

契約ネットモデル[10]、オブジェクト指向モデル[11]などの協調問題解決モデルが提案され、それらのモデルに基づくシステムが開発されている。協調問題解決では部分問題を扱う複数の知識源あるいはエージェントが通信を行いながら、協調して問題を解く。一般に知識源とエージェントはほぼ同じ意味で使われ、最近ではエージェントという呼び方が広く用いられている。本論文では、黒板モデルでは慣例に従い知識源と呼び、それ以外ではエージェントと呼ぶことにする。

分散協調問題解決は基本的には分散環境における問題解決を目的としている。従って問題全体を依存度の少ない比較的粒度の大きな問題に分割し、それらを扱うエージェントが自律的にかつ少ない通信のもとで解く必要がある[4]。これに対し本研究が対象としている並列協調問題解決は、部分問題間の依存性が比較的大きい問題を、並列計算機上で解くことを目的としており、分散環境ほどエージェントの自律性は要求されない。

従来のモデルの中で部分問題間の依存関係が強い問題に適していると考えられるのは黒板モデルであり[4]、これに基づく設計システムもいくつか報告されている[7]、[8]、[9]。そこで、ここでは黒板モデルをベースに本研究の課題について検討する。

黒板モデルは複数の知識源が一種のデータベースとしてのブラックボードを共有し、データ駆動で起動される知識源が協調的に推論を進める。ただし、各知識源の動作順序を動的に決定するスケジューリング機構を設けることにより協調動作を実現するのが普通である。

黒板モデルの最大の利点は現実の設計過程との対応がよいことである。すなわち、部分問題を扱う複数の知識源が協調して黒板上に設計解を生成していくというモデルは、複数の設計者が部分問題を担当し協力しあって設計解を生成するという設計過程を素直にモデル化したものである。

従来の黒板システムのほとんどは逐次処理計算機上で実現されたものだが、最近、並列計算機の上で実現する研究がなされるようになった[12]、[13]。しかし、並列黒板システムの実現は容易ではない。

黒板システムの並列化でまず問題になるのがスケジューリングである。複数のプロセッサを効率よく活用するためには、非常に複雑なスケジューリングが必要となる。また、スケジューリング処理が並列化におけるボトルネックになる可能性がある[14]。従って陽にスケジューリングを行うのではなく、より単純で並列処理に適した協調動作が要求される。

並列化可能な処理としては、黒板上のデータ操作の並列化、複数の知識源の並列動作、各知識源内の問題解決の並列実行、スケジューリングと知識源の処理の並列実行などがある[15]。しかし、これらの並列化手法のみでは高い並列度を得るのは難しい[16]。並列化による効率向上の方法としては基本的に、複数の処理の並列実行、データのパイプライン処理、大量のデータの並列処理があり[14]、効率的な並列処理を実現するにはこれらを効果的に活用する必要がある。特に、大きな並列度を得るには大量のデータの並列処理が有効であるとの報告がある[16]。

データの整合性の維持も大きな課題である。これには黒板上のデータ全体の整合性、問題解決のコンテクストの整合性、データ値保持の整合性（評価中に値が変化しない）などが含まれる[14]。解を合成した後に特定の知識源で整合性をチェックする方式ではなく、より自然で効率的な整合性維持機構が求められている。

さらに、複数のプロセッサへの処理の割り当ても難しい問題である。単に各知識源をそれぞれプロセッサに割り当てるのみでは、大きな並列度は得にくい。黒板へのアクセスがボトルネックになる可能性もある。また、対象とする計算機が共有メモリ型のマルチプロセッサの場合には問題が少ないが、分散メモリ型のマルチプロセッサの場合には、黒板をいずれかひとつのプロセッサに割り当てるか、複数のより小さな黒板に分割して各プロセッサに割り当てるかの方法があるが[15]、効率的な実現は難しい。

以上のように黒板モデルは設計過程モデルとの対応はよいものの、並列協調問題解決システムとしての課題は多い。本研究の目的はこれらの課題を解決した新しい並列協調問題解決を実現することにある。

3. 並列協調問題解決モデル

3.1 構成と動作

ここでは前述の課題を解決するため、モジュール性のよい知識表現、単純で効率のよい協調動作、大量のデータによる並列処理、解の整合性の維持、並列計算機上での効率よい実現等が可能な並列協調問題解決モデルを提案する。

まず、基本構成について考える。システム構築を容易にするには、知識表現レベルの構成をできるだけ現実の設計過程との対応のよいものとすべきである。特に、知識の維持、管理を容易にするには、知識のモジュール化が重要である。そこでここでは、まず設計で用いられる知識を設計対象に関する知識と設計方法に関する知識に分離し[17]、さらに設計方法に関する知識を部分問題毎にモジュール化する。

設計対象に関する知識は設計対象モデル[18),19)]として表現する。対象モデルは設計対象の形状や構造、種々の属性、部品間の関係、動作などを表現するもので、これらの表現に適したオブジェクト指向に基づく表現を採用する[20]。ただし、属性間あるいは部品間の関係や設計対象が満たすべき条件を宣言的に記述可能とするために、制約表現機能を導入する[21]。要求仕様も設計対象に関する制約と見なすことができる。

一方、設計方法に関する知識は、設計手順や設計式、部品の選択法や設計対象の属性値の決定法など、ヒューリスティクスを含む設計の手続き的知識である。設計方法に関する知識は部分問題毎に異なり、それぞれエージェント単位にモジュール化して管理し、問題解決に利用する。各エージェントは設計対象モデルを共有するが、それぞれ設計対象の特定の側面のみを扱うため、ひとつのエージェントが参照するのは設計対象モデルの一部に限られる。

一般にエージェント間の協調動作はエージェント間で直接通信を行うか、ひとつの推論制御機構がエージェントを一括管理する方式が用いられる。しかし前者はエージェントが互いに他のエージェントを知る必要があり、後者は推論制御機構が全てのエージェントを知る必要があるため、知識のモジュール性を損なう恐れがある。

そこで我々は部分問題間の依存関係は設計対象の共有により発生することに着目し、知識のモジュール性を損なうことなく、設計対象に存在する制約を積極的に利用して協調動作を実現する方式を提案する。図1は本方式の基本構成である。すなわち、設計対象モデルを制約充足機能を持つ能動的なオブジェクトを用いて表現することにより、全体の整合性を維持しながら、対象モデルを介した制約伝播により、明示的な通信を行うことなくエージェント間で協調動作を行うことができる。制約に基づく協調動作の詳細は

後述する。

本システムでは、エージェントが設計途上で少しずつ出力する設計データ（中間解）を合成し、対象モデルの制約に違反する解候補を枝刈りしながら全体として設計が進むという形式と、他のエージェントが生成した中間解あるいはそれと関連のあるデータに基づいて自分の解を生成するという形式が協調動作の基本である。

次に並列動作について考える。設計解は多数の設計データの集合である。一般に解候補は複数存在するため、設計データは複数の値をとり得る。設計データの組み合わせによって表わされる解の状態をコンテクストと呼ぶ。大きな並列度を得るには大量のデータを扱うことが必要がある。そこで本システムでは複数のコンテクストを並列に処理するため、オブジェクトにマルチコンテクスト管理機能を持たせる。

また、本システムではエージェントを並列処理の単位とするのではなく、対象モデルを構成するオブジェクトやエージェントの内部の粒度の小さな処理ひとつひとつを並列処理の単位とする。それらの処理単位はそれぞれひとつのプロセスで表現され、プロセスの数が並列計算機の持つプロセッサ数に比較して十分大きなものとなるようにする。そして各プロセッサの処理ができるだけ均等になるようにそれらを分散させ、プロッセサの平均稼働率の向上を図る。

3.2 設計対象モデル

設計では、あらかじめ得られている設計対象に関する知識は対象一般に関する知識のみで、要求仕様を満足する具体的かつ詳細な設計対象モデルを生成するのが設計の目的である。そこで、対象モデルの一般的知識をクラス宣言で記述しておき、設計時に必要によりオブジェクト（インスタンス）を生成し、設計データをそのスロット（インスタンス変数）に記憶する[21]。

クラス宣言では属性値を記憶するためのスロットや、ひとつのスロットあるいは複数のスロット間の制約を宣言する。オブジェクトに記述する制約はそのオブジェクトの持つスロットとそのオブジェクトのスロットに格納されている（正確にはポインタが格納される）オブジェクトの制約のみが参照可能である[21]。

図２（ａ）に、マイクロプロセッサのクラス宣言の例を示す。“slot”以下にマイクロプロセッサの属性を表現するスロットが宣言され、“constraint”以下にスロット値の組み合わせ禁止（データパスが１バスのときは制御方式として命令先読みは採用できないことを nogood で指定）や方程式（性能の計算式）などの制約が記述されている。クラス宣言はトランスレータにより実行形式に変換される。

本システムにおけるオブジェクトはそれ自身が制約充足機能を持つ[21]、[22]。すなわち、オブジェクトのスロットに値が設定された時に制約を評価し、充足処理を実行する。そしてこの制約充足処理は整合性維持や協調動作に役立つ。

オブジェクトの持つスロットの一部は外部に公開される。ただし、対象モデルの見方、すなわち参照するオブジェクトやスロットはエージェントにより異なるのが普通である。本システムでは各エージェントに見えるスロットの集合を指定することにより、それらスロットの変化をエージェントに知らせるデモン機能を提供する。ただし、その指定を明示的にする必要をなくすため、トランスレータが自動的にエージェントのルールを解析して得る機能を提供する。

各オブジェクトは並列に実行可能なプロセスの集合で表現され、オブジェクトへのメッセージは並行に

解釈実行される。従って同一オブジェクトの同一スロットへのアクセスを除いて、対象モデルには並列にアクセスできる。またオブジェクトの提供するマルチコンテクスト機能により、ひとつのスロットについて複数の値を扱うことができる。

設計対象モデルは従来の黒板上のデータを構造化し、制約充足など能動的な機能を持たせたものと見なすことができる。黒板システムでは階層化した黒板がよく用いられるが[6]、本システムではこれを対象モデルを表現するオブジェクトの階層で表わすことができる。しかもその構造を問題解決時に動的に生成、変更することが可能であり、動的に設計対象モデルを生成する設計問題では有効である。

3.3 エージェント

本研究はもともと部分問題毎、すなわちエージェント毎に異なる問題解決方式を用いることのできるシステムを目指しており、基本的にはエージェントの推論方式が何であるかは問わない。しかし現在のシステムではエージェントの知識はプロダクションルールを用いて表現し、前向きのプロダクションシステムとして動作するものとしている。ルールは図2（b）のように、エージェント単位にモジュール化して記述され、トランスレータにより実行形式に変換される。

エージェントでは対象モデルから送られてくるスロット値のデータが条件部にマッチしたルールが発火される。ここで、複数の解候補を並列に取り扱うため、従来のプロダクションシステムでは不可欠であった競合解消処理を行わず、発火可能なルールは全て同時に発火する方式を採用する。そのかわり、複数のルールが発火した後の状態をそれぞれ別のコンテクストとして扱うことにより、整合性を保つことができる。

コンテクストを維持するため、エージェントがオブジェクトにスロット値を設定する場合には、そのコンテクストを指定して値を設定する必要がある。本システムでは、例えば

```
if
    class(Obj, マイクロプロセッサ),
    slot(Obj, {スロット１, X}), X = 1
then
    set_slot(Obj, {スロット２, 2});
```

という、クラスがマイクロプロセッサのオブジェクトのスロット１の値が１ならば、スロット２に値２を設定することを表わすルールの実行部を、トランスレータにより

```
set_slots(Obj, [{スロット１, 1},
                {スロット２, 2}])
```

という形に変換する。そして、実行時には、新たに設定するスロット値のみでなく条件部で指定されたスロットとの組み合わせが設定される。これにより、オブジェクトにおいて正しくコンテクスト管理される。

エージェントは対象モデルの変化に応じて推論を行う。具体的にはオブジェクトから変化のあった（新たに設定された）スロット値のデータを受け取り、（一般には別の）スロット値を返すという処理である。従って、ひとつのルールの実行毎に対象モデル上で中間解が組み合わされ、制約評価されるため、制約に違反する解候補は早期に枝刈りされる。

以上のように対象モデルとエージェントとのインタフェースはデータフローを基本とする。一般に設計データは複数のエージェントを経由して詳細化が進むが、複数の解候補を次々に処理することにより，パ

イプライン並列の効果を得ることができる。

4. 制約充足

4.1 整合性維持

オブジェクトは制約を評価し、充足処理を実行する。制約評価には受動的な評価と能動的な評価がある。受動的な評価とは設定されたスロット値が制約を満足するかどうかをチェックするもので、能動的な評価とは制約評価によってそれまで未定であったスロット値が決定されるものを言う。例えば x + y = 10 という制約が存在する場合、受動的な評価とは x と y の両方の値が与えられた後その和が 10 かどうかを調べるもので、能動的な評価とは どちらか一方の値が与えられた時にその和が 10 になるように他方の値を決定するものである。

制約の受動的評価は整合性の維持のための基本処理である。前述のように解の整合性を保つことは並列処理では非常に重要である。整合性維持は、制約を受動的に評価し、制約を満足する無矛盾なオブジェクトの状態を維持することである。すなわち、複数のエージェントが異なるスロットに値を設定した場合、それらのスロット間の制約を評価し、制約を満足しない組み合わせ（コンテクスト）は生成されない。従って全体として整合性のある解のみを保持するとともに、組み合わせ爆発の回避に利用できる。

オブジェクト外に整合性維持機構を設けて解の整合性をチェックする方式は、一旦解候補を生成した後に整合性チェックを行うため効率がよくないうえ、設計では多数の解候補を扱うため処理のボトルネックとなる可能性がある。これに対し本システムでは解候補を表わすオブジェクト自身が整合性を保つ機能を持つため、そのような問題を回避できる。

4.2 制約伝播

制約を能動的に評価することにより、あるエージェントの処理の影響を対象モデルを通して他のエージェントに伝播することができる。例えば図３のようにエージェント１の出力したスロットａの値とａ・ｃ間の制約によりスロットｃの値が求まり、それを他のエージェントに伝播することができる。これは単一オブジェクト内の制約の例であるが、複数のオブジェクト間の制約を利用することも可能である。

このような対象モデルを経由した制約伝播を利用することにより、あるエージェントが生成した中間解の影響を他のエージェントに間接的に伝播することができる。従って本システムでは共有データを持たないエージェント間でも直接通信を行うことなく協調動作が可能である。また、制約伝播の方向は実行時に動的に決定されるから、エージェントの動作順序にかかわらず情報交換ができる。

本システムでは能動的な評価によって値が一意に決定可能な制約についてのみ能動的な評価を行う。例えば x + y = 10 という等式表現された制約は能動的評価可能であるが、x + y > 10 という不等式で表現された制約は受動的評価しか行わない。

5. マルチコンテクスト管理

5.1 マルチコンテクスト

本システムにおけるコンテクストとは、スロット値の組み合わせによって表わされるオブジェクトの状

態のことである。一般にコンテクストの変化は木構造で表現することができ[23]、図4はその例である。スロット値が設定されるにつれ、木構造が成長していく。

一般に設計が進むにつれ、コンテクストは増加する。しかし、図4のように制約に違反するコンテクストは削除することができるため、制約を満足する整合性のあるコンテクストは単調に増加するわけではない。逆に言うと制約に違反するコンテクストをできるかぎり早期に刈り取ることが処理の効率上重要である。

また、本研究が対象とする問題はヒューリスティクスを利用して試行錯誤的に解を求める必要があるうえ、各エージェントが非同期に並列処理を行うため、オブジェクトの生成やスロット値の設定の順序は非決定的である。従って図4のように、分岐したコンテクストがその後同じ状態になることがあるが、この場合には冗長な処理をなくすため、両者を統合化して同一のコンテクストとして扱える機能が必要である。

複数の仮説の組み合わせを扱う機能を有するシステムとしてＡＴＭＳ[24]が有名である。各スロットの値を仮定（assumption）と見なしてＡＴＭＳを適用すれば、整合性維持機能を持った、冗長性のないマルチコンテクスト管理を実現できる。しかし、スロット値ひとつひとつを仮定としたのではノード数が膨大になるうえ、制約に違反する組み合わせのみでなく、コンテクストの形成に意味のない組み合わせ（例えば“スロット１＝１”と“スロット１＝２”の組み合わせ）も生成しないように大量のnogoodを与える必要があり、処理が繁雑になる。ＡＴＭＳを利用して多重世界を表現する手法も提案されているが[25]、本システムにおけるひとつのコンテクストをひとつの世界として扱うと、世界の数が膨大になり処理効率が悪化する。従って、オブジェクトの構造を利用した単純で効率のよいマルチコンテクスト処理が望まれる。

5.2 マルチコンテクスト・オブジェクト

本システムではオブジェクト自身にマルチコンテクストを扱う機能を持たせる。すなわち、オブジェクトにスロットの組み合わせを管理するノードを設け、各ノードがスロット値の組み合わせを管理する。

ノードは図5のように、ひとつのスロットを扱うノードを先頭として、先行するノードが扱うスロットが後続するノードが扱うスロットの部分集合とするように接続されたノード網を形成する。ＡＴＭＳのノード網と似た形をしているが、各ノードはスロット値の組み合わせでなくスロットの組み合わせであり、ひとつのノードが複数のデータを扱う。ノード網はオブジェクト（インスタンス）の生成と同時に生成する。ノードはひとつのプロセスで表現される。従ってノード網は複数のプロセスの集合であり、それらのプロセスは並列に実行される。

制約はそれが参照するスロットの組み合わせを扱うノードに記憶される。そしてノード自身が制約充足処理を実行し、制約を満足するもののみを記憶する。従って制約を満足しないスロット値の組み合わせは削除される（生成されない）。

一般にノードは複数の先行ノードから複数のデータを受け取り、それらを組み合わせて、そのノードが扱うデータを生成する。図5では最後尾のノードに制約を満足するスロット値の組み合わせが記憶されている。

ところが、スロットの全ての組み合わせを扱うのではノード数が膨大になり、大きな処理時間を要する。そこでシステムにとって意味を持たない、すなわち不要なスロットの組み合わせをなくすことにより、ノード数を減少させる。

まず、オブジェクトにとって必要なのはそのオブジェクトに与えられた制約が参照するスロットの組み合わせである。また、エージェントから見たコンテクストはそのエージェントが参照するスロット値の組み合わせのみで決定される。従って、制約が参照するスロットの組み合わせとエージェントによって参照されるスロットの組み合わせのみを生成すれば十分である[22]。本システムではトランスレータによる変換時にオブジェクトの約とエージェントのルールを解析することにより必要なノードを得、ノード網を最適化する。

スロットに他のオブジェクト（正確にはそのポインタ）が格納され、そのオブジェクトのスロットを制約が参照する場合には、それを扱うノードも生成する。そして格納されているオブジェクトのスロットに値が設定された時に、それを制約が記述されているオブジェクトに報告することによりスロット値の組み合わせを生成する。

エージェントのルールの条件部で参照されているスロットの組み合わせを扱うノードにはデモンが設定されており、制約を満足する新しいデータを記憶する時にデモンが起動され、データはエージェントに送られる。

5.3 動作

エージェントと対象モデルを表現するオブジェクトのノードの動作を図６を例に説明する。対象モデルを４つのエージェントが囲んでいる。オブジェクトのノード網は最適化されている。なお、図ではルールの表現を簡略化している。

まず要求仕様が与えられるとそれを参照するルールを持つエージェント１とエージェント２が並列に起動される。それらのエージェントには条件部が同じルールが複数存在するがこれらも競合解消なしに並列に発火する。これによりオブジェクトのスロットａを扱うノードに“１，２”、スロットｂを扱うノードに“３，４”が記憶される。そして、それらのデータは後続するスロットａとｂの組み合わせを扱うノードに送られ、そこで組み合わせデータを生成する。ところがそのノードには制約が与えられているため、制約を満足するデータのみが記憶され、後続のノードに送られる。

スロットａ、ｂ、ｃの組み合わせを扱うノードでは、送られてきたａ、ｂの組み合わせデータを用いて制約を能動的に評価することによりスロットｃの値を算出でき、ａ、ｂ、ｃの組み合わせデータを生成する。ここで、算出されたｃの値はスロットｃを扱うノードに送られる。そしてｃはエージェント３のルールの条件部によって参照されているため、そのデータはデモンによりエージェント３に送られる。

エージェント３ではふたつのルールが発火され、スロットｄの値を設定する。ただし、前述のように実際にはスロットｃとｄの組み合わせデータを設定する。そしてｃ、ｄを扱うノードで制約を満足しないデータは削除される。最後に、スロットａ、ｂ、ｃ、ｄを扱うノードで組み合わせデータが生成され、エージェント４に送られ、出力される。

上記の組み合わせ処理で、複数の先行ノードのデータが同一のスロットを含む場合はそれが同じ値の時のみ組み合わせを生成できる。例えば図６でスロットａ、ｂ、ｃの組み合わせデータは３つ存在するが、ｃ、ｄの組み合わせデータは“５、６”のみであるため、それらを組み合わせたａ、ｂ、ｃ、ｄのデータはスロットｃの値が５のものしか許されない。

本システムでは動的にオブジェクトに制約を追加することができる。その場合、制約はそれが参照するスロットの組み合わせを扱うノードに設定される。もしノードに記憶されているデータのうち、追加され

た制約に違反するデータが存在すればそれを削除するとともに、それを部分集合としているデータの削除を後続ノードに命じる。そして削除命令が伝播されていき、最終的に制約を満足しないデータが全て削除される。なお、現在は制約を動的に削除することは許していない。

ところで、ノード網はプロダクションシステムにおけるＲＥＴＥネットワークの機能の一部を実現している。従ってエージェントにおけるルールの条件部のマッチング処理は非常に簡単な処理となる。また、ルールの競合解消も不要であるから、効率のよい推論が可能となっている。

6. 設計問題への応用

以上提案した並列協調問題解決方式を用いた設計システムのプロトタイプを試作した[27]。対象はマイクロプロッセサの命令セットや性能仕様を入力とし、レジスタ・トランスファ・レベルのブロック構造とデータパスおよび動作記述を出力する問題である。ただし試作したプロトタイプは方式評価を第一の目的としているため、現実の問題を簡単化し、少数のオブジェクト及びルールで動作可能なシステムとした。

まず、問題分析を行い、設計全体を設計方針決定、動作設計、ブロック抽出、データパス設計、タイミング設計の５つの部分問題に分割した。設計方針決定はマイクロプロセッサのアーキテクチャの基本構造を決定する部分、動作設計は命令セットを解析して基本的な動作を決定する部分である。ブロック抽出は命令の実行に必要な機能ブロックを抽出する部分、データパス設計はデータの流れる経路を設計する部分である。タイミング設計は命令フェッチや命令動作に必要なタイミングをマシンサイクル単位で設計する部分である。

試作したシステムには上記の各部分問題を扱う５つのエージェントと、設計結果を出力するエージェントが存在する。設計結果出力エージェントを除き、各エージェントのほとんどのルールは中間解の生成ルールである。ルール数は全部で約６０である。ただし、条件部が同じで異なる部分解候補を生成するルールは同種のルールとしてひとつと数えた。

我々はこのシステムをＧＨＣ[28]をベースとした並列論理型言語ＫＬ１[29]を用いて開発した。設計対象を表現するオブジェクトのクラス定義と、ｉｆ－ｔｈｅｎ形式で記述された各エージェントのルールはトランスレータにより、ＫＬ１プログラムに変換され、それをコンパイルすることにより実行可能となる。扱う制約は簡単な方程式と不等式および組み合わせ禁止の制約であり、方程式のみが能動的評価可能である。

このプロトタイプを開発することにより、本方式を用いれば、並列処理を特に意識することなく、設計対象をオブジェクトで、設計方法に関する知識をルールで記述するのみで、並列協調設計システムを容易に構築できることを確認した。

ここで扱った問題は中間解の候補が多数存在し、単純に組み合わせるとおよそ千個の解候補が生成される。しかしそれらのうち、要求仕様および設計対象に関する制約を満足する解は数個である。本方式では制約を利用して解候補の枝刈りを行いながら協調的に設計を進めるため、各ノードが扱う組み合わせの数は数十個程度に押えられている。従って、部分問題間で設計対象を共有し、部分解の候補が多数存在するものの要求仕様を満足する設計解は少数である問題に対しては、本論文で提案した並列協調問題解決方式が有効であることが確認できた。

なお、問題によっては制約による枝刈りのみでは組み合わせ爆発に対応できない可能性もあるため、解候補に優先順位を設けて、まず優先順位の高い解候補のみを生成し、それで解が得られない場合にのみ優

先順位の低い解候補を生成する機能も導入している。

本システムは分散メモリ型のマルチプロセッサ計算機、Multi-PSI [29] 上で動く。現在のシステムでは静的な負荷分散を採用した。まず、各エージェントを各プロセッサに割り振り、次にノードを各プロセッサがほぼ同数のノードを扱うように分散させる。

現在のシステムの台数効果はプロセッサ４台の時１台の時の約２倍、８台の時約２．７倍である。台数効果が期待したほどよくないのは、処理の前半は各プロッセサの稼働率が１００％近いものの、処理の後半ではデータの流れによって処理量の多いノードとそうでないノードが生じ、プロセッサの稼働率に大きなばらつきが生じるためである。台数効果を向上させるには、問題の規模を大きくするとともに、動的負荷分散を含め、より効率のよい負荷分散方式の検討が必要である。

7. おわりに

以上設計問題を対象に、部分問題間の依存性が大きい問題に適した並列協調問題解決方式を提案した。この方式では、制約充足機能とマルチコンテクスト管理機能を持つオブジェクトにより表現した対象モデルを共有する複数のエージェントが並列に問題解決を実行する。

そして、対象モデルを経由した間接的な制約伝播により、エージェント間の通信を陽に記述することなく協調的な動作を実現することができる。また、スロットの組み合わせを扱うノード網を用いたマルチコンテクスト機能により、エージェントのルールではコンテクストを意識することなく、多数の解候補を並列に取り扱い可能である。

また、この方式に基づいた並列協調設計システムのプロトタイプを並列論理型言語を用いて、分散メモリ型のマルチプロセッサ計算機上に試作し、本方式の有効性を確認した。

今後の課題としては、制約を仮説として扱うことやコンテクストによって異なる制約を扱うなど、機能面での充実を図るとともに、負荷分散方式について検討を深め、より効率のよいシステムを実現することなどがある。

謝辞

本研究の機会を与えていただくとともにご指導いただいた、ＩＣＯＴの淵一博所長、生駒憲治部長代理、新田克己室長、および日頃御討論いただくＩＣＯＴ第七研究室の皆さんに感謝する。

参考文献

1) Tong, C. : Toward an Engineering Science of Knowledge-Based Design, Artificial Intelligence in Engineering, vol.2, no.3, pp.133-166 (1987).

2) Klein, M. and Lu, Stephen C.-Y. : Conflict Resolution in Cooperative Design, Artificial Intelligence in Engineering, vol.4, no.4, pp.168-180 (1989).

3) Bond, A. H. and Gasser, L. (eds.) : Readings in Distributed Artificial Intelligence, Morgan Kaufmann, California (1988).

4) 小野典彦：分散協調問題解決, 淵一博（監修）：知識プログラミング, pp.41-65, 共立 (1988).

5) Hayes-Roth, B. : A Blackboard Architecture for Control, Artificial Intelligence, vol.26, pp.251-321 (1985).
6) Nii, H. P. : Blackboard Systems : The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures, The AI Magazine, vol.7, no.2, pp.38-53 (1986).
7) Rehak, D. R. et al. : Architecture of an Integrated Knowledge Based Environment for Structural Enginnering Applications, in Gero, J. (ed.) : Knowledge Engineering in Computer Aided Design, pp.89-117, North-Holland, Amsterdam (1985).
8) Bushnell, M. L. and Director, S. W. : ULYSSES - a Knowledge-Based VLSI Design Environment, Artificial Intelligence in Engineering, vol.2, No.1, pp.33-41 (1987).
9) Temme, K-H. and Nitsche, A. : Chip-Architecture Planning: an Expert System Approach, in Gero, J. S. : Artificial Intelligence in Engeneering: Design, pp.137-161, Elsevior, Amsterdam (1988).
10) Smith, R. G. and Davis, R. : Frameworks for Cooperation in Distributed Problem Solving, IEEE Trans. on Systems, Man and Cybernetics, SMC-11-1, pp.61-70 (1981).
11) Yonezawa, A and Tokoro, M (eds.) : Object-Oriented Concurrent Programming, MIT Press, Massachusets (1987).
12) Engelmore, R. and Morgan, T. (eds.) : Blackboard Systems, Addison Wesley, Wokingham (1988).
13) Jagannathan, V. et al. (eds.) : Blackboard Architectures and Applications, Academic Press, Boston (1989).
14) Nii, H. P. et al. : Frameworks for Concurrent Problem Solving: A Report on CAGE and POLIGON, in 12), pp.475-501 (1988).
15) Corkill, D. D. : Design Alternatives for Parallel and Distributed Blackboard Systems, in 13), pp.99-136 (1989).
16) Rice, J. et al. : See How They Run... The Architecture and Peformance of Two Concurrent Blackboard Systems, in 13), pp.153-178 (1989).
17) 永井保夫ほか：設計問題向けツールアーキテクチャ，人工知能学会誌，vol.4, no.3, pp.297-303 (1989).
18) 上野晴樹：知識工学入門，6.1 対象モデル，オーム社 (1985).
19) Ohsuga, S. : Conceptual Design of CAD Systems Involving Knowledge Base, in Gero, J. (ed.) : Knowledge Engineering in Computer Aided Design, pp.29-88, North-Holland, Amsterdam (1985).
20) 木村文彦：オブジェクト指向によるＣＡＤ／ＣＡＭのためのモデリングとデータベース，情報処理，vol.29, no.4, pp368-373 (1988).
21) 横山孝典，佐塚秀人：制約に基づくオブジェクト指向知識表現システム，情報処理学会論文誌，vol.31, No.1, pp.68-75 (1990).
22) 横山孝典ほか：並列協調問題解決のための対象モデル表現方式，情報処理学会第41回全国大会予稿集，1K-6 (1990).
23) Walters, J. R. and Nielsen N. R. : Crafting Knowledge-Based Systems, Chapter 15 Knowledge Crafting with Multiple Contexts, John Wiley & Sons, New York (1988).
24) de Kleer, J. : An Assumption-based TMS, Artificial Intelligence, Vol.28, pp.127-162 (1986).
25) Morris, P. H. and Nado, R. A. : Representing Actions with an Assumption-Based Truth Maintenance System, AAAI-86 Proceedings, pp.13-17 (1986).
26) Forgy, C. L. : RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, Artificial Intelligence, vol.19, pp.17-38 (1982).
27) 小野昌之ほか：設計向き並列協調問題解決システムの提案，情報処理学会第41回全国大会予稿集，

1K-8 (1990).

28) Ueda, K. : Guarded Horn Clauses, ICOT Technical Report, TR-103 (1985).

29) Uchida, S. et al. : Research and Development of the Parallel Inference System in the Intermediate Stage of the FGCS Project, Proceedings of the Fifth Generation Computer Systems, pp.16-36 (1988).

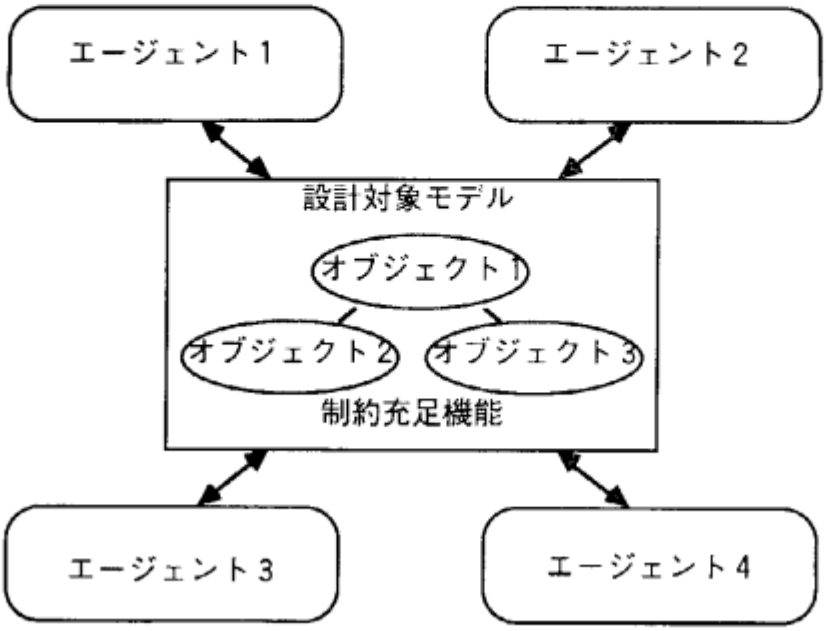

図1　並列協調問題解決モデル
Fig.1　Parallel cooperative problem solving model

```
class マイクロプロセッサ has
  slot
    演算器, 内部転送方式, 命令動作,
    制御方式, データパス, 性能, マシンサイクル,
    平均ステップ数, ・・・・・・;
  constraint
    nogood（［内部転送方式, 制御方式］,
                ['1バス', '命令先読み']）,
    性能＝1／（平均ステップ数 * マシンサイクル）,
     ・・・・・・;
end.
```

(a) オブジェクトのクラス宣言の例

```
agent データパス設計 has
  rule
    if      class（Obj, マイクロプロセッサ）,
            slot（Obj,  {命令動作, B}  ）
    then    generate_data_path(B, D),
            set_slot（Obj,  {データパス, D}  ）;
     ・・・・・・;
end.
```

(b) エージェントのルールの記述例

図2 知識表現
Fig. 2 Knowledge Representation

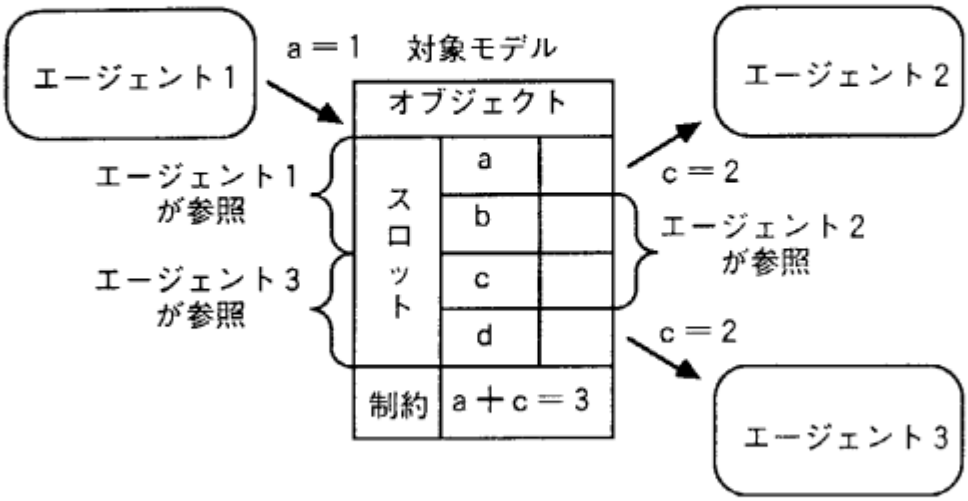

図3　対象 モデルを経由したエージェント間の制約伝播
Fig. 3　Constraint propagation　among agents

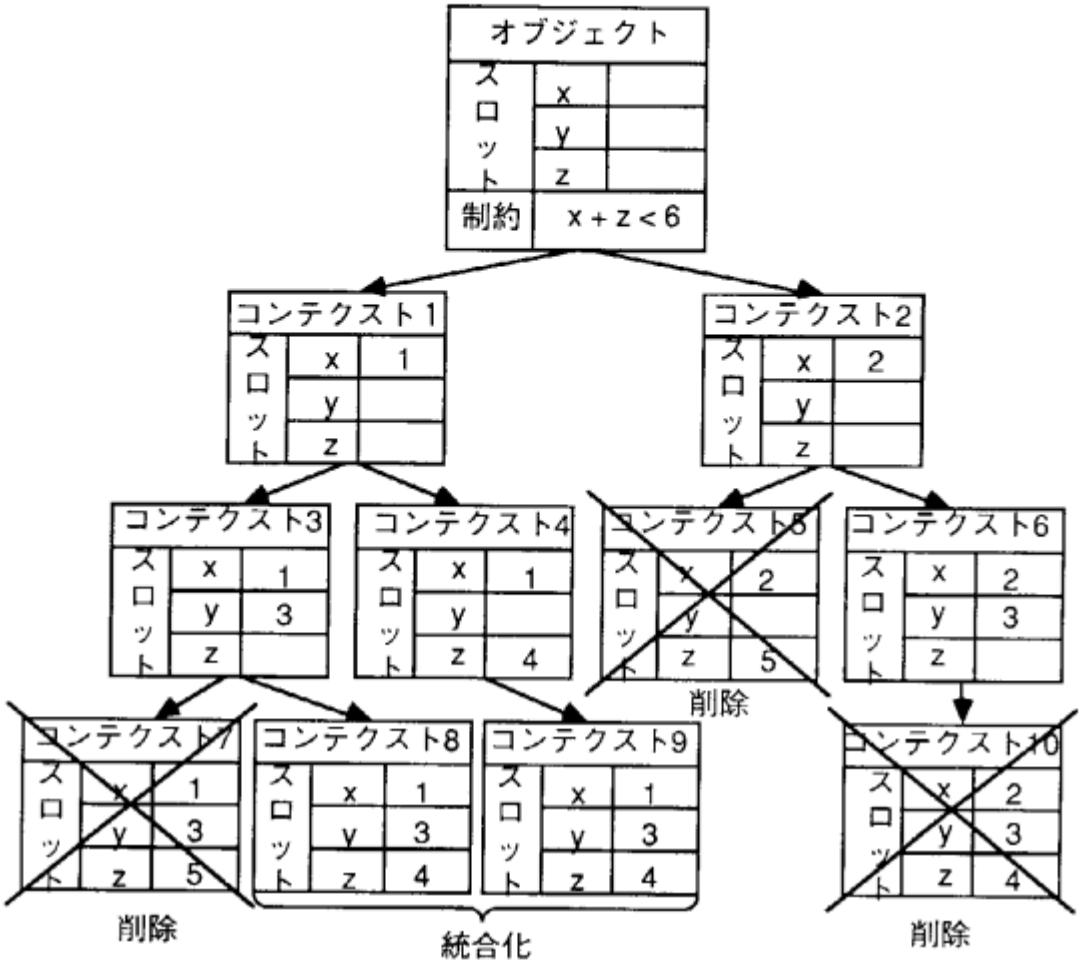

図4　木構造表現したコンテクストの例
Fig. 4　An example of context tree

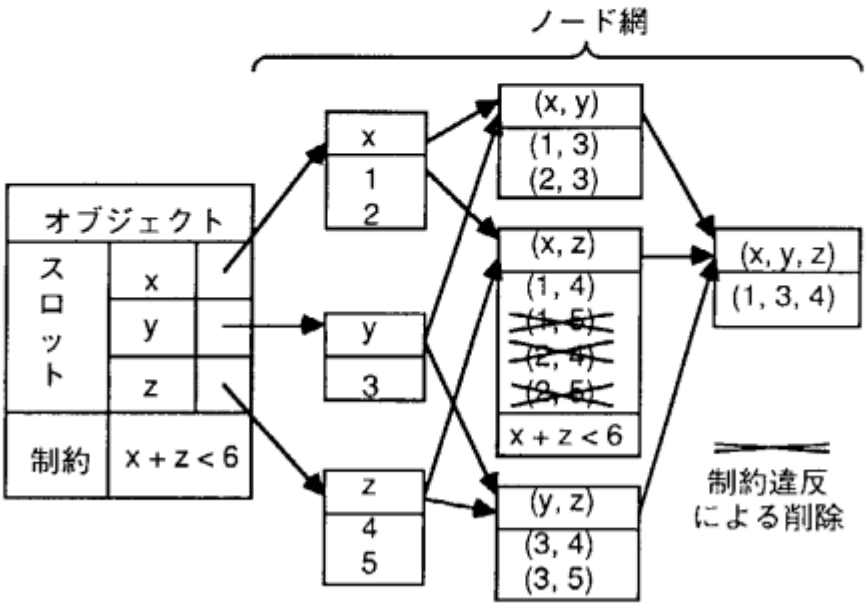

図5　マルチコンテクストのためのノード網
Fig. 5　Node network for multiple context

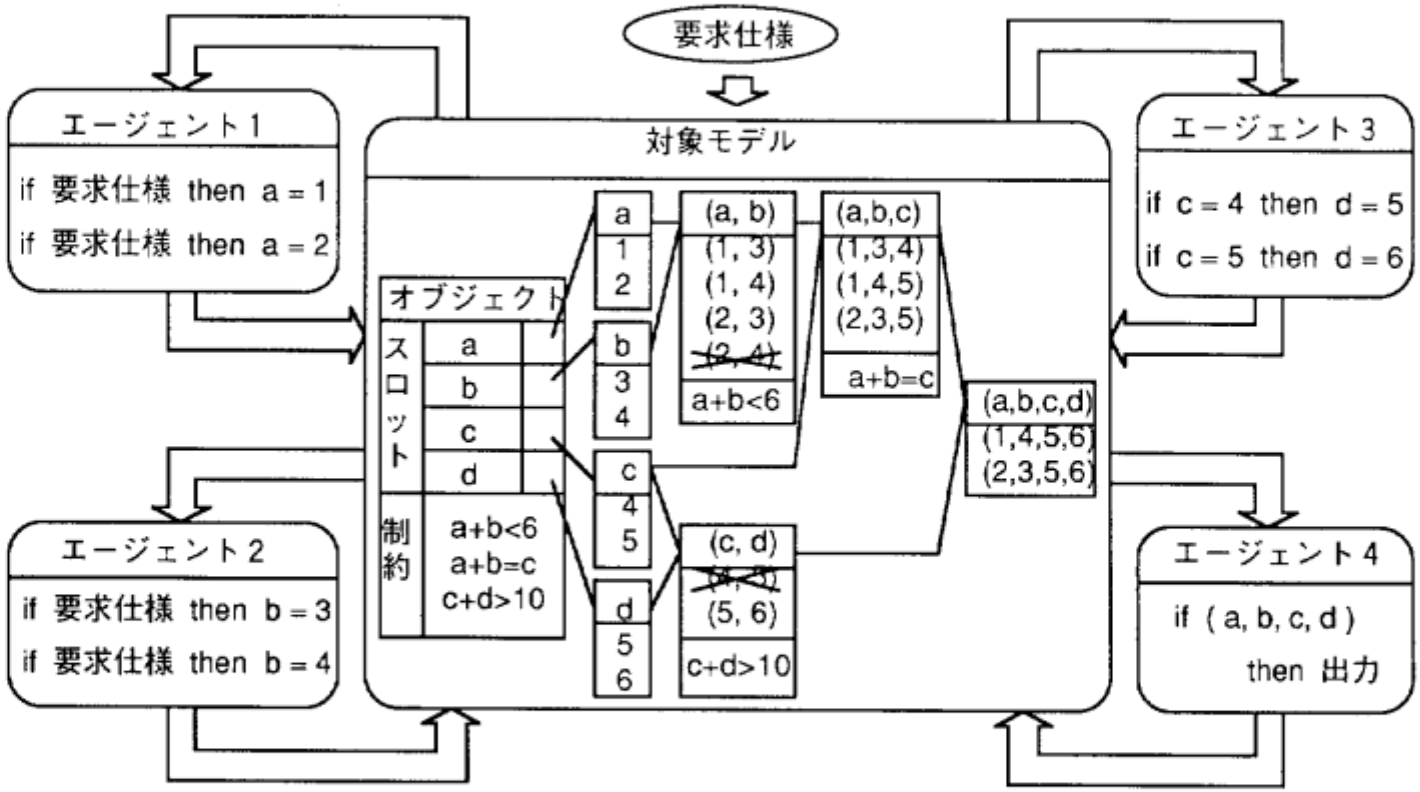

図6 エージェントとノード網の動作
Fig. 6 Agents and node network