# マルチエージェントシステムにおける協調メカニズムの研究

1997年3月

桑原 和宏

# マルチエージェントシステムにおける
# 協調メカニズムの研究

1997年3月

桑原 和宏

# 序文

マルチエージェントシステムは自律的に動作するエージェントとよばれるプログラム群から構成されるシステムで，新しい分散システム構築のモデルとして注目を集めている．マルチエージェントシステムではエージェント間の協調をいかに実現するかが大きな課題となっている．本論文ではこのような背景のもとマルチエージェントシステムにおける協調メカニズムにおいて筆者が行なってきた以下の研究をまとめたものである．

- マルチステージネゴシエーションプロトコル
- 市場モデルに基づく分散資源割当: 均衡的アプローチ
- 協調プロトコル記述言語 AgenTalk

第１章において本研究の背景を概説するとともに本研究の目的を明確にする．

第２章ではマルチエージェントシステムにおける協調メカニズム研究状況の概要を述べ，本論文の位置付けを明確化する．

第３,４章ではエージェント間の協調を実現するためのプロトコル (協調プロトコル) の一例としてエージェント間にまたがる制約を満足した資源割当を実現するマルチステージネゴシエーションを取り上げる．まず，第３章においてマルチステージネゴシエーションで扱っている問題の定式化を与え，その中で複数のゴール間の競合を検出するための手法を提案する．さらに第４章でマルチステージネゴシエーションにおける探索戦略 (３フェーズプロトコル) の評価を行なう．

第５章では市場モデルに基づく分散資源割当の試みを述べる．マルチステージネゴシエーションでは資源の割当をするかしないかを選択する問題を扱っていた．これに対し，マルチエージェントシステムの中に市場の構造を導入することによって，資源

の量を扱えるようにする．それと同時にエージェント間の限られたメッセージの交換のみでエージェント間の協調を実現することを目指す．ここではその第一段階として均衡的アプローチと呼ぶ市場モデルに基づくアプローチを提案し，シミュレーション実験結果を中心に述べる．

第6章ではマルチエージェントシステムにおける種々の協調を実現するための“協調プロトコル”を記述するための言語 AgenTalk を提案する．AgenTalk ではプロトコル記述にオブジェクト指向言語に見られるような継承機能を導入し，プロトコルの段階的な定義を実現しているのが特徴である．さらに AgenTalk のアーキテクチャに基づいた協調プロトコルのメタレベル制御機構についても言及する．

第7章では結論と今後の課題について述べる．また，最後に付録として AgenTalk の仕様概要を述べる．

# 目次

# 図一覧

# 表一覧

# 第 1 章

# 序論

近年，計算機技術の発展により高性能な計算機が安価に入手できるようになってきた．さらに，ATM(Asynchronous Transfer Mode) 網，移動体通信など，ネットワーク技術の進歩に伴い，多くの計算機がネットワークを通して連携できるようになっている．特に多くの計算機がネットワークで結ばれた分散システムでは外部環境に対してオープンなシステムとなり，システム全体を一箇所に集中して，設計，運用することが不可能となりつつある．このような背景のもと，新たな分散システムの設計手法が求められている．

このとき新しいシステムの設計手法として考えられるのがマルチエージェントシステムの考え方である．すなわち，自律的に振舞うエージェントを複数集めてシステム全体を構成しようとするものである．マルチエージェントシステムの考え方を採用することにより，システム全体を集中して設計するのではなく，個々に設計されたエージェントと呼ばれるプログラム群の集まりとしてシステム全体を構成することになる．マルチエージェントシステムとして実現することにより，種々のエージェントを必要に応じて追加することで，システムの柔軟性を実現したり，また，一つのエージェントが仮に故障したとしても，他のエージェントによって機能を肩代わりするなど，システムの頑強性を達成できることが期待される．

このようなマルチエージェントシステムを実現する上での中心的な課題の一つとしていかにエージェント間で協調 (coordination) を達成するかということがあげられる．“協調” という言葉はいろいろな意味に用いられるが，ここでは複数のエージェントがそれぞれの自分の目標を効率よく達成するために (共に) 行動するという意味としてとらえる．このときエージェントの目標が共通の場合もあれば，必ずしも共通で

ない場合も考えられる．

エージェントの目標が共通であるときマルチエージェントにおける協調は分散協調問題解決と呼ばれる分野で研究されてきた [2, 11]．そこではエージェントが達成する共通の目標を複数の副問題に分割し，個々の副問題を解いた結果を統合して最終的な結果を導く．いかに副問題に分割するか，それぞれの副問題の問題解決をいかに制御するか，また，副問題の結果をいかに統合するかが課題となる．また，エージェントの目標が異なる場合はエージェント間で交渉し，必要に応じて目標間の競合解消を行い，それぞれの目標を達成することが必要になる．まず第2章ではマルチエージェントシステムにおける協調メカニズムの研究状況を概説する．

このようなエージェント間の協調を実現するために，一般にエージェントはお互いにメッセージを交換する．ここではエージェント間の協調を達成するための一連のメッセージの交換のプロセスを協調プロセス (coordination process) とよび，協調プロセスにおけるエージェント間の (アプリケーション層の) プロトコルを協調プロトコル (coordination protocol) と呼ぶことにする．

従来，分散人工知能の分野を中心として多くの協調プロトコルが提案されてきた．その代表的なものは契約ネットプロトコル (Contract Net Protocol)[9, 55, 56] である．契約ネットプロトコルは分散協調問題解決においてタスク割当を行なうために提案されたものである．基本的な契約ネットプロトコルでは，タスク公告 (*Task Announcement*)，入札 (*Bid*)，落札 (*Award*) などのメッセージによってタスクをエージェント間に分配する．他のエージェントに割り当てたいタスクを持つエージェントはマネージャ (manager) となりタスク公告を他のエージェントに対して放送する．タスク公告を受けとったエージェントは提示されたタスクを自分のところで実行しようとする場合は入札をマネージャに対して送る．マネージャは送られてきた入札の中から最も適当と判断するエージェントに対して落札を送る．落札を受け取ったエージェントはコントラクタ (contractor) としてタスクを実行することになる．このようなメッセージの交換を通してタスクの分配が実現される．

契約ネットプロトコルではエージェントでは各自の観点からタスク割当の情報を評価し，最終的に得られるタスクの割当は，マネージャとコントラクタが相互に選択 (mutual selection) したものになるのが特徴といえる．いわばエージェント間の情報交換の結果，タスク割当に関するエージェント間の協調が実現されたと考えることができる．



契約ネットプロトコルは極めて基本的なプロトコルであり，多くの問題領域に応用されている．例えば，分散システムにおけるタスクスケジューリング [44]，トラックの配車問題 [53] などがあげられる．

契約ネットプロトコルにおいてはエージェントの出す入札，落札を取り消すことができないため，グローバルな制約が存在する場合，グローバルな制約を満足するようにタスクの割当を行なうことができないという問題がある．これに対して，グローバルな制約のもとで複数のエージェントが協調して資源の割当を行なうためのマルチステージネゴシエーションプロトコルが提案されている．マルチステージネゴシエーションでは，通信網において通信路が故障した場合の代替パスの探索が最初の例題として取り上げられた [7]．そこでは，通信網は複数の地域に分割され，各々のエージェントが担当地域内の通信路の割当に責任を持つ．どのエージェントも通信網全体の情報を持たず，自分の担当区域内の局所的な資源の割当を他のエージェントに通信する．そのとき他エージェントにおける局所的な資源の割当が自分の担当区域内の資源割当に及ぼす影響を互いに通知しあうことによって全体として制約を満足した資源の割当を行なうのが特徴となっている．

このマルチステージネゴシエーションは契約ネットプロトコルの一種の拡張・一般化として考えることができる．契約ネットプロトコルでは一回のタスク公告，入札，落札というメッセージの交換でタスクの割当を行なっているのに対し，マルチステージネゴシエーションでは，局所的な資源割当の影響に関して交換した情報をもとに，必要に応じて資源割当をやり直す．

本論文の第3章では，まず，このマルチステージネゴシエーションで扱われている問題の定式化を行う．続いて，グローバルな目標間の競合関係，すなわち，制約が強過ぎて，すべての制約を満足することができないという競合関係を検出する手法を述べる．さらに第4章ではマルチステージネゴシエーションにおいて制約を満足するための解を探索する探索戦略を述べ，その評価を行う．

マルチステージネゴシエーションの扱った問題では資源の割当について，割り当てるか，割り当てないかのゼロ一の問題を扱っている．しかし，通信路の割り当てという例題を考えると資源の量を扱えることが必要になる．そこで次に資源割当を資源の売買という形で実現する市場モデルの導入を試みる．

人間の経済活動はもともと情報・制御が分散している中で，大規模な資源割当を行なっていると考えることができ，マルチエージェントシステムのモデルと親和性が高

い．特に競争市場の考え方においては，協調のためのエージェント間の明示的な通信無しに資源割当が実現できることが期待できる．

そこで第5章では，そのような試みの一つとして (競争) 市場のモデルをマルチエージェントシステムにおける資源割当に応用した例を述べる．ここでは均衡的アプローチ (equilibratory approach) と呼ぶ手法を提案し，そのシミュレーション実験の結果について考察する．

さらに，このような協調メカニズムを用いて実際にマルチエージェントシステムを構築するためにはエージェント間のプロトコルを記述するための言語が望まれる．特に契約ネットプロトコル，マルチステージネゴシエーションプロトコルを初めとする協調プロトコルは，応用領域には独立なプロトコルである．これらのプロトコルを用いて実際にアプリケーションを構築するためには，各アプリケーションに特化したプロトコルとすることが必要となる．

第6章では，協調プロトコルを記述するための言語 AgenTalk を提案する．この言語ではオブジェクト指向言語に見られるような継承機能を導入し，プロトコルの段階的な定義を実現しているのが特徴である．また，プロトコル定義のうちアプリケーション依存の部分を容易にカスタマイズするためのインタフェースを設けている．

最後に第7章で本論文の内容を総括するとともに今後の課題を述べる．最後に付録として第6章で提案する AgenTalk の仕様概要を述べる．

# 第 2 章

# マルチエージェントシステムにおける協調メカニズムの研究状況

## 2.1 はじめに

本章の目的はマルチエージェントシステムにおける協調メカニズムに関する研究状況を概観し，本論文の研究内容の位置付けを明確にすることである．

マルチエージェントシステムの研究は分散人工知能 (distributed artificial intelligence) の研究から出発したといえる．分散人工知能は人間社会をはじめとする集団の諸活動を自律的に判断する複数の計算主体 (エージェント) の相互作用としてモデル化し，工学的応用を図る研究分野である [21]．初期の分散人工知能の研究の多くは，全てのエージェントが一つの目標を対象とする分散問題解決 (distributed problem solving) を対象としていた．その後，エージェントの自律性が意識されるようになり，必ずしもエージェントの目標が共通ではない場合が取り上げられ，マルチエージェントシステムの研究として発展している [2]．

マルチエージェントシステムの研究では，情報を一箇所に集中することができない場合，ないし，集中することが得策でない場合を対象としている．情報を分散させ，並列性をあげることで性能を向上させるのが目的ではない．情報を一箇所に集めることができない場合として例えば次のような場合が考えられる．

- 通信の遅延が問題になる場合:

  制御信号の伝達には光速度の限界が存在する．例えば，高速ネットワークにおいて A,B の 2 地点間を結ぶ通信路でパケットの送出制御を行なうことを考えよ

う．仮にA点で処理能力が限界になったためA点へ向けたパケット送出を止める命令をB点に送るとする．A,B地点間の距離が離れているとするとA点から出た制御命令がB点に到達する前にすでに多数のパケットがA,Bの2地点間を結ぶ通信路上に存在していることになる．この場合B点に制御命令が届いてからパケットの送出を停止するのでは時間的に間にあわないことになる．さらに極端な例では，地球より遠く離れた宇宙探索機から送られてくるセンサ情報を地球で受けてから一つ一つ命令を送るのでは，通信遅延から制御命令が間に合わない場合が考えられる．このような場合は通信遅延が問題となり情報を一箇所に集めて集中的に制御することができない．

- エージェントのプライバシー，セキュリティーが問題になる場合:

  すべての情報を他のエージェントに対して公開することがプライバシー，セキュリティーを確保する上で問題となる場合がある．例えば会合スケジューリングシステムでは，参加者のすべてのスケジュール，好みを他のエージェントに公開するのは好ましくないであろう．適当なスケジュールを決定するのに必要な情報のみを他のエージェントに公開したいという要求が出てこよう．このような場合は通信の遅延というよりも問題の性質から情報を一箇所に集めることができない．

このような要求条件のもとでは，情報を一箇所に集めて，問題を解くことができないことになる．そこで，複数の情報処理主体 (これをエージェントと呼ぶ) の間で情報のやりとりを行ない，問題を解くことが必要となる．すなわち，マルチエージェントシステムの適用分野となる．

このようなマルチエージェントシステムにおける課題は次の二つにわけて考えることができる [20].

1. 協調問題解決: 複数のエージェントが共通の目標を持つ場合に，エージェント間で協力し，組織を構成し，共通の目標を達成する．

2. 交渉と均衡化: 複数のエージェントあるいはエージェントからなる組織群が独立の目標を持つ場合に，それらエージェント (組織) 群の目標間に生じる競合を解消し，全体として好ましいバランスを維持しながら，それぞれの目標を達成する．



以下，マルチエージェントシステムにおける協調メカニズムにおける従来の研究を二つの観点から述べ，本研究の位置付けを明確にする．

## 2.2 協調問題解決

従来から複数の計算機が通信ネットワークによって接続された分散システムの研究が行なわれている．分散システムには，並列かつ非同期的に計算を実行することによる性能の向上，環境の変化にしたがってシステムを段階的に機能を拡張できる拡張性，システムの一部が故障しても，他の部分で代用することによって機能を継続可能とする耐故障性などの潜在的な特長がある．このような分散システムを通信ネットワークを介して情報を交換する複数のプロセスとしてモデル化し，複数プロセスを効果的に協調動作させるための分散アルゴリズムも研究されている [23].

分散システムにおける情報に関してLesserらは情報の正確性とエージェントの自律性を二つの軸として次のような分類を提案している [40]．従来の分散システムは“完全に正確で，ほぼ自律的” (Completely Accurate, Nearly Autonomous – CA/NA)，すなわち，各エージェントは独立に副問題を解くことができ，他のエージェントの情報が必要な場合にも正確な情報が得られるとの仮定のもとで議論を展開してきたといえる (表2.1)．これに対して分散協調問題解決においてエージェントが不完全な情報のもとで副問題を解き，部分解をお互いに交換することが必要になってくる．上記の分類では“(完全ではないが) 機能的に正確で，協調的” (Functionally Accurate, Cooperative – FA/C) といえる．従来の分散システムがエージェント間でデータが矛盾することを許容しなかったことと大きく異なっている．すなわち，すべての情報を持たない中でエージェントは必要に応じて他のエージェントとメッセージを交換し，自分の行動を決定していくことが必要となる．このときの協調メカニズムとして結果共有，タスク共有，分散探索の考え方が提案されている [21].

### 2.2.1 結果共有

マルチエージェントシステムにおいてはエージェントが問題すべての情報を得ることができない．そこで，エージェントが副問題の中間結果をお互いに持ち寄ることで最終的な目標とする解に到達しようとする結果共有 (result sharing) の考え方が提案されている [56]．音声認識を行なう分散型Hearsay-II [42] や，複数の基地局が協力し

表 2.1: 分散システムの形態

| | 完全に正確 (completely accurate) | 機能的に正確 (functionally accurate) |
|---|---|---|
| ほぼ自律的 (nearly autonomous) | CA/NA | (FA/NA) |
| 協調的 (cooperative) | (CA/C) | FA/C |

て乗物を追跡する DVMT (Distributed Vehicle Monitoring Testbed) [41] がその代表的な例である．これらのシステムでは，エージェントが問題のすべての情報を集めることができず，局所的情報をもとにエージェントが判断を下す必要がある．例えば，DVMT では，エージェントは複数の地域に分けられた担当区域のセンサ情報を有するものの，他の地域のセンサ情報をもたない．そのため，エージェントの間で適切に情報を交換し，複数の担当区域をまたがる乗物の軌跡を求める必要がある．

結果共有の考え方では，複数のエージェントが整合 (coherency) のとれた判断を行なうか否かが効率に大きく影響してくる．例えば，解が存在すると思われる方向に全エージェントが協力して探索を行なわれなければ，効率よく探索を行なうことができない．そこで PGP (Partial Global Planning) の考え方が提案されている [10]．PGP では，エージェントは相互に問題解決の計画の概略を送る．エージェントは他のエージェントの計画を考慮に入れ，システム全体の動きを把握する．必要に応じて各自の計画を修正し，システム全体の方向と自分の局所的な計画が合うようにする．修正した計画の概略を再度他のエージェントに送信する．図 2.1 は DVMT におけるセンサ情報の例である．この図では多少の重複を持ちながら，四つの地域に分割された各地域にエージェントが割り当てられている．図中のノードがセンサからの信号を表し，ノード間を結ぶ線がセンサからの信号をもとに解析された乗物の軌跡を表している．ここでは，エージェント 1 では $d_4, \ldots, d_{12}$ と $d'_1, \ldots, d'_5$ の信号があり，エージェント 2 では $d_{10}, \ldots, d_{15}$ の信号があり，エージェント 3 では $d_1, \ldots, d_6$ の信号がある．このときエージェント 3 は $d_1, \ldots, d_6$ を処理していることを自エージェントにお



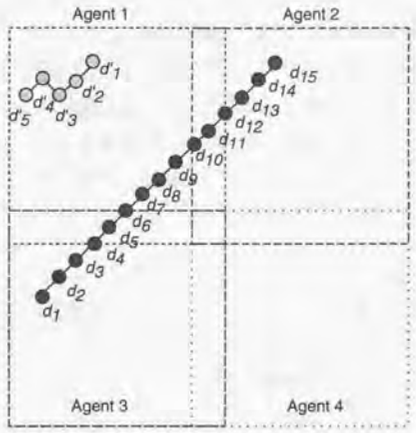



図 2.1: DVMT におけるセンサ情報の例 [10]

ける局所的な計画としてエージェント 1 に対して送信する．エージェント 1 では仮に $d'_1, \ldots, d'_5$ を先に処理していたとしても，エージェント 3 から局所的な計画を受けとると $d_4, \ldots, d_{12}$ の処理を優先する．これは，エージェント 1 にとっては $d'_1, \ldots, d'_5$ がノイズであることも考えられ，これらの信号よりもエージェント 3 から来た信号につながる $d_4, \ldots, d_{12}$ を先に処理した方が効率がよいと考えられるからである．このようにして系全体の効率をあげることができる．すなわち，エージェントの自律性と系全体の効率性という，相異なる要求を満足する枠組を提供している．

### 2.2.2　タスク共有

一方，問題解決に要する負荷を複数のエージェントに分割して割り当てる協調の方式はタスク共有 (task sharing) と呼ばれている [56]．タスク共有という呼称は，複数のエージェントが一つのタスクを “共有し”，エージェント間の合意の下にサブタスクに分割して分担するところから付けられている．タスク共有では副問題をエージェント群にいかに割り当てるかが問題となる．先に述べた結果共有の考え方とタスク共有の考え方はお互いに独立のものというよりもむしろ相補的なものである．一つの問題

に対して，タスク共有の考え方に基づいてサブタスクに分割，割り当てられたあと中間結果をまとめる際に結果共有の考え方を適用するということも考えられる．

第 1 章で述べた契約ネットプロトコル [55] はタスク共有を実現するためのプロトコルと考えることができる．契約ネットプロトコルは人間社会における交渉 (negotiation) 過程をエージェント間の合意形成に応用したもので，タスク (あるいはサブタスク) を分配するエージェントはマネージャ (manager) と呼ばれ，タスクの分配においてはタスク提示 (*Task Announcement*) を行なう．タスクを処理するエージェントはコントラクタ (contractor) と呼ばれ，提示されたタスク (あるいはサブタスク) の中から各自の評価基準で選択し入札 (*Bid*) を送る．マネージャは多くの入札の中から，各自の評価基準で選択し落札 (*Award*) を送る．このように契約ネットプロトコルで最終的に得られるタスクの割当は，マネジャとコントラクタが相互に選択 (mutual selection) した結果となっているのが特徴である．

契約ネットプロトコルは，多数のセンサを用いて広範囲を監視制御する分散センシングシステムやローカルエリアネットワークの負荷分散 [44] などに適用されている．また，本論文でとりあげるマルチステージネゴシエーションは契約ネットプロトコルを拡張し，複数の契約間に大域的な制約が存在する場合を対象としたものであると考えることができる．

### 2.2.3　分散探索

また，マルチエージェントシステムにおける問題解決を問題空間の探索として定式化した分散探索の考え方が提案されている [20]．分散探索は高速化を目的とする並列探索とは異なり，エージェント間の情報に矛盾がある場合や，副問題の間に大域制約が存在する場合が対象となっている．分散探索として定式化することにより，エージェント間で交換する情報を規定することができる．分散探索として具体的にはエージェント間の情報の無矛盾性 (consistency) を維持する手法，エージェント間の状態間に制約がある場合に制約を満足するような状態を見つける手法が提案されている．

#### エージェント間の情報の無矛盾性の維持

エージェントがあるエージェントから受信した情報をもとに $P$ を演繹したとしよう．同時に他のエージェントから受信した情報をもとに $\neg P$ を演繹したとするとエージェント間の情報に何らかの矛盾があることになる．



分散 ATMS (Assumption-based Truth Maintenance System)[45] の手法では，エージェントはそれぞれ ATMS の機能を持ち，エージェント間で交換するメッセージにおいて情報の根拠となる仮説 (assumption) を付加する．受信したエージェントはすでに持っている情報と矛盾が発見したときは矛盾が生じる原因となる仮説の組合せを無効集合 (nogood) として検出し，他のエージェントに送信する．エージェント間で無効集合を共有することによって共有情報の無矛盾性を維持する．

分散 ATMS ではすべての無矛盾な仮説の組合せを保持するのに対し，分散 TMS (Truth Maintenance System)[18] の手法では，エージェント間の情報で無矛盾な仮説の組になるように維持する手法を提案している．TMS と同様に情報に IN, OUT のラベルをつける．このときエージェント間の情報の無矛盾性の度合を 1) 各エージェントの内部で情報が矛盾している場合 (矛盾: inconsistent)，2) 各エージェントの内部では無矛盾 (局所無矛盾: local consistent)，3) エージェント内部の情報とエージェント間の共有情報は無矛盾 (局所共有無矛盾: local-and-shared consistent)，4) 全エージェントの情報が無矛盾 (大域無矛盾: global consistent) の 4 レベルに分類している．マルチエージェントの枠組では局所共有無矛盾が一つの妥当なレベルであるが，その局所共有無矛盾を満足するメカニズムを提案している．

#### 分散制約充足

さらに，分散探索の一つとしてエージェント間の大域制約を満足させる分散制約充足問題 (distributed constraint satisfaction problem) が提案されている．解法としては，非同期バックトラックアルゴリズム [63] などが提案されている．これはエージェントの局所変数のとる値の間に満足すべき制約があるときに，制約を満足するような値の組合せを見つけるアルゴリズムである．特に非同期バックトラックアルゴリズムは解が存在すれば必ず解を見つけるという完全性 (completeness) を満たすアルゴリズムとなっている．詳しくは第 4 章で述べるが，基本的にはエージェントが局所解を他のエージェントに通知し，かつ矛盾となる無効集合 (nogood set) を通知することを繰り返す．

本論文ではマルチステージネゴシエーションの枠組 [7] を題材としてとりあげる．これは第 1 章でも述べたように大域的な制約を満足しつつ資源割当を行なうための枠組である．契約ネットプロトコルの拡張，一般化という観点からはタスク共有を行なうための手法としても考えられ，また，大域的な制約を満足させる割当を実現するという観点からは一種の分散探索の手法としても考えることができる．本論文ではマルチステージネゴシエーションで扱っている問題における探索空間を明確にし，探索空間の定式化に基づいてエージェント間で交換する情報を明確にする．詳細情報すべてを他のエージェントに送るのではなく，いかに情報を抽象化して送るかが問題となり，ここでは送るべき情報を明確にする．マルチステージネゴシエーションでは複数のグローバルなゴールが存在するときに大域的な制約を満足させる割当を求めるのが特徴である．グローバルゴールを考慮に入れて定式化した探索空間に基づいてエージェントが交換すべき情報を規定することによりグローバルゴール間の競合が正しく認識できるようにする．さらに分散制約充足アルゴリズムとの定量的性能の比較を行ない，分散探索という観点からの比較評価を行なう．ここでは探索空間の定式化に基づいて交換する情報を規定しているので，制約が強過ぎるときに複数のゴールのうちどのゴールをあきらめれば残りのゴールについて解を求められるかを明らかにできるという特徴を持っている．

## 2.3　交渉と均衡化

協調問題解決ではエージェントはある一つの問題を複数のエージェントで協調して解くことが対象となっていた．さらにエージェントが個々の目標を持っている場合，エージェント間の目標の競合を解消し，システム全体として適当な状態を維持することが必要になってくる．

### 2.3.1　ゲーム理論的手法

エージェントの合理的な行動基準を与える数理的手法としてゲーム理論がある．ゲームは，意思決定主体としてのプレイヤ (エージェントに相当)，プレイヤのとる行動，プレイヤの行動に対して得られる利得 (実数値で表現) でモデル化される．この



ゲーム理論を用いて，エージェント間の交渉プロトコルが研究されている．

ゲーム理論では一般に利得行列 (payoff matrix) でエージェントの行動の選択肢とその行動をとったとき得られる利得 (payoff) を定義する．交渉を行なうとき，各エージェントは，自分の行動と自分が受け入れることのできる交渉相手の行動の組 (offer group) を，交渉相手に提示する．ここで，すべてのエージェントの提示案に含まれる行動の組が存在すれば，それを交渉の妥結点 (deal) とする．そのような行動の組が複数ある場合には，適当な調停者が一意に決定する．この交渉モデルを用いてエージェントの合理性 (rationality) に種々の定義を与えることにより，協調的行動を実現するメカニズムが提案されている [52].

また，ゲーム理論はマルチエージェントプランニングにも応用されている．例えばエージェントの目標が必ずしも一致しない場合の共同プランの作成の可能性について検討されている [65]．図 2.2 はこれを模式的に表現したものである．図の各点が状態に対応しており，領域 $G_1, G_2$ はそれぞれエージェント 1,2 の目標が満足される状態の集合を表す．$s$ は初期状態を表し，エージェントが単独で目標を満足させる時のコストは $s$ と目標を表す領域との距離で示されている．三重線は両エージェントの共同プランを表している．ここでは，エージェントの利得を，プランの実行によって達成される目標の価値とプランの実行に要するコストの差として定義する．

協調的状況では図 2.2 (1) に示すように 2 エージェントの目標領域の共通部分が初期状態に近く，エージェント間で協力するのが有利であることがわかる．一方，競合的状況では 2 エージェントの目標領域間に共通部分がなく，共同プランが成立しない．そこで共同プランを次のように拡張する．まず初期状態 $s$ からある状態 $t$ に移行する混合共同プラン (mixed joint plan) $J$ を考える．混合共同プランとは共同プランのどの部分をどのエージェントが実行するかを確率的に決定するものである．ここでは，エージェントの能力は同じで，どちらのエージェントもプランの任意の部分を実行できると仮定している．さらに状態 $t$ において確率 $q$ でどちらか一方の目標を選択することにする．このような共同プランは $(t, J, q)$ で表され，図 2.2 (2) に示すようになる．これによりエージェント間で目標が一致しない場合 (例えば，2 エージェントが一つの荷物を異なる場所に運ぼうとする) においても，共同作業が成立する (途中まで協力して荷物を運ぶと双方の期待効用が増大する) ことになる．なお，このように拡張された共同プランは協調的状況にも適用可能である ($t \in G_1 \cap G_2$ とすればよい) ので，この考え方に基づいて協調的状況，競合的状況共に適用可能な統合交渉プロト

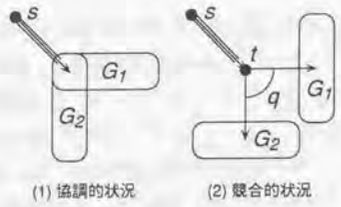



図 2.2: 協調的状況と競合的状況 [65]

コル (unified negotiation protocol) が提案されている [65]．このようにゲーム理論は独自の効用を持つエージェントの合理的行動基準を提供してくれる．

### 2.3.2　市場モデルの応用

人間の経済活動は極めて大規模な資源割当を非集中制御のもとで行なっていると考えることができる．経済活動における人間の行動は分散人工知能において自律的なエージェントを実現する上で有用なモデルを提供してくれる．経済活動の中でも，市場 (market) は分散資源割当を実現する具体的なメカニズムを提供している．ここでエージェントは独自の効用 (関数) を持ち，エージェントは自分の効用を最大にするように振舞う．競争市場においてはいわゆる “神のみえざる手” にしたがってシステム全体としてパレート最適な均衡状態が実現できる．

さらに市場において，財・サービス (以下では財で代表させる) に価格がつけられ，貨幣と交換される．市場に参加する経済主体 (消費者，生産者) は財につけられた価格を基に何をどれだけ購入し，また，何をどれだけ生産するかを決める．価格を用いることにより経済主体の選好をはじめとして種々の情報を極めて簡潔な形式に抽象化して表現している．これにより，エージェント間で交換する情報を押えつつ，適当なシステムの振舞いを実現できることが期待される．

このような市場の考え方を応用して，計算機上に仮想的な計算的市場 (computational market) を構築し，種々の資源の割当を集中制御ではなく分散制御のもとで実現することが試みられている [3]．以下では，動的な分散資源割当へ応用した研究例を述べる．ここにおける資源は主としてプロセッサの CPU 時間であり，分散システ



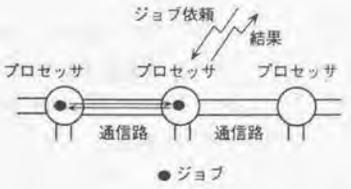



図 2.3: プロセッサ間のジョブの移動

ムにおける CPU 時間という資源をいかにジョブに割り当て，負荷分散させるかが課題となっている．これらの研究では競争市場の持つ理論的な性質を活用するというより，分散して資源割当を行なうためのヒューリスティクスのよりどころを市場モデルに求めている．

#### ジョブの移動による負荷分散

この例は複数のプロセッサが通信路で結ばれており，プロセッサに到着したジョブがプロセッサ間を移動し実行されることでプロセッサ間の負荷分散を実現しようとするものである [12]．この枠組では CPU 時間の他にプロセッサ間を移動するときの通信路の帯域が財として扱われる．各プロセッサが CPU 時間と他のプロセッサへジョブを移動するための通信路の帯域を提供する生産者となる．また，システムに入力される各ジョブがそれらのサービスの消費者となる．ジョブには資金があらかじめ与えられ，これを用いてプロセッサからジョブ実行に必要な CPU 時間を購入し，また，他のプロセッサに移動するときはプロセッサ間の通信路の帯域を購入することになる．他のプロセッサに移動したときでも最終的な結果はジョブが到着したプロセッサに戻るという前提がおかれている．したがって，別のプロセッサへ移動するときは結果を返すためにもとのプロセッサに戻るための資金をとっておく必要がある．また，ここでは価格の概念を導入していることにより，CPU 時間と通信路の帯域という異なる資源を同列に扱うことが可能になっている (図 2.3)．

CPU 時間と通信路の帯域の価格は競売 (auction) によって決められ，プロセッサは自分の利潤を最大にするように CPU 時間と通信路の帯域を販売する．これに対し

ジョブの目的は与えられた資金を用いてジョブを完了させることである．

ここでは (1) できるだけ安くジョブを実行する価格優先方式，及び，(2) ジョブの終了時間をできるだけ短くするサービス時間優先方式が検討されている．二つの方式案の混合も考えられるが，その性質は 2 方式の中間的なものになる．

9 台のプロセッサが 3 × 3 の格子状に接続されたプロセッサネットワークを対象にシミュレーションを行った結果，いずれの方式を採用した場合でも，従来手法に比べ，ジョブの平均待ち時間が同等か，多くの場合，短くなることがわかっている．このシミュレーションでは個々のジョブの完了に要する時間は指数分布にしたがうとし，また，最初に与える資金の額は同じという仮定をおいている．その結果，(1) 価格優先方式では，ジョブは CPU 時間が安いプロセッサへ移動しようとする．そのため，通信路の価格が上昇する．最初に与える資金の額は同じなので，処理時間の短いジョブは，単位時間あたりの資金が多くなり，通信路の帯域の競売に勝ってプロセッサ間を移動する傾向が見られる．(2) サービス時間優先方式では，プロセッサ間の移動によるジョブ終了の遅れを少なくするために，ジョブは同じプロセッサに留まろうとする．処理時間の長いジョブは，単位時間あたりの資金が少ないため，CPU 時間の競売に負けることが多くなり，プロセッサ間を多く移動する傾向が見られる．

このように，ジョブの入札の方針を変更することで移動するジョブの種類を変えることができる．このことはエージェントの局所的な方式を制御することでシステム全体の振舞いを制御できる可能性を示している．

#### 資金量の制御による負荷分散

LAN でつながれたワークステーションの CPU 時間の割当において市場モデルを導入した Spawn と呼ばれるシステムが提案されている [59]．ここではワークステーション (プロセッサ) が生産者 (売り手) に対応し，自分の余っている CPU 時間を売る．応用プログラム (ジョブ) が消費者 (買い手) に対応し，必要な CPU 時間を売り手より買い求める．

前述のジョブの移動による負荷分散とは異なり，Spawn ではジョブがいくつかのサブタスクに分解され，並行に実行される．サブタスクは必要に応じてさらにいくつかのサブタスクに分解される．したがって，ジョブはツリー状の階層構成をとるサブタスク群によって実行されることになる．各サブタスクごとに (アプリケーション) マネージャが存在し，マネージャは必要に応じて自分のタスクをサブタスクに分解し，サブタスクを実行するマネージャを新たに起動する．



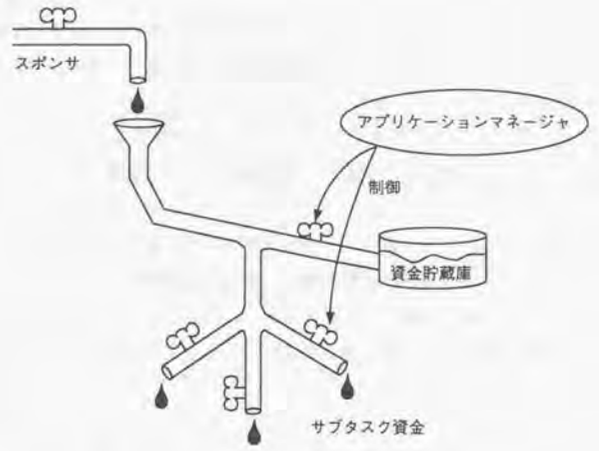



図 2.4: Spawn における資金の流れ [59]

ジョブの実行に際してユーザがスポンサとなってサブタスク階層のツリーの根に相当するトップレベルのマネージャに資金を与える．各マネージャは手持ちの資金を優先度にしたがって自分のサブタスク間に分配する (図 2.4)．また，サブタスクに渡されずに余った資金は一旦資金貯蔵庫に蓄えられ将来の使用に備えられる．

売り手のプロセッサは空きの CPU 時間を販売するために競売を行なう．Spawn の競売では入札者は他の入札者の入札価格を知ることができず，また，一番高い価格を提示した入札者が入札に勝つが，支払う価格は 2 番目に高い入札価格とする Vickrey の入札方式 [58] を採用している．この方式では入札者は自分にとって本来の価値に見合った金額を入札するようになる．例えば，入札に勝てない危険をおかして本来の価値より少ない価格で入札するとしよう．もし，入札に勝てなかったとすると，他の入札者が支払う金額を少なくする可能を増やすだけになり，自分の得にはならない．したがって，入札に勝てなくなる危険をおかしてまで，本来の価値より少ない価格で

入札する意味はないことになる．さらに，この方式は，他で良く見られる競売の方式 (公開の場で，他により高い価格を提示する入札者が現れなくなるまで，入札価格を徐々にあげていく方式) と比べて，少ない通信のオーバヘッドで同じような価格に決まることが知られている．

Spawn ではシミュレータを用いたシミュレーションだけではなく，性能の異なるワークステーションが接続された実際の LAN 環境において実験が行なわれている．例題としてモンテカルロシミュレーションを行なうプログラムが取り上げられ，与える資金の量を制御することによりジョブやサブタスク間の優先度を制御できることが示されている．性能の違うプロセッサが存在する異機種環境においてはプロセッサ間の性能の差は CPU 時間の価格の差に現われる．このように価格を導入することにより，プロセッサ間の性能の違いを考慮することなくジョブ / サブタスクの優先度を与えることができる．

また，緊急性の高いジョブが発生したとしよう．その緊急ジョブに多くの資金を与えて実行させると CPU 時間の価格が一時的に上昇し，今まで実行していたジョブは CPU 時間を得ることができず，緊急ジョブが優先的に実行されるようになる．緊急ジョブが終了すると，CPU 時間の価格がもとに戻り，もともと実行していたジョブが CPU 時間を使用できるようになる．このように動的なジョブの発生に対しても柔軟に対応できる．

このように価格という概念を導入することにより，種々の要素が価格という一つの尺度で表現できるという利点が生まれる．前述の研究では価格が競売によって決定される．したがって競売機構のオーバヘッド (処理時間，処理能力) が問題になる場合が予想される．それに対し，本研究では売り手が価格を自律的に決定する機構を想定し，競売のオーバヘッドをなくすアプローチ (ここでは “均衡的アプローチ” と呼んでいる) を提案し，その性質をシミュレーション実験によって明らかにする．

## 2.4　協調プロトコル記述言語

以上述べたような協調メカニズムを用いて，実際にシステムを構築するにあたってはエージェントのインタラクションにおけるプロトコル (協調プロトコル) を記述できるような言語が望まれる．



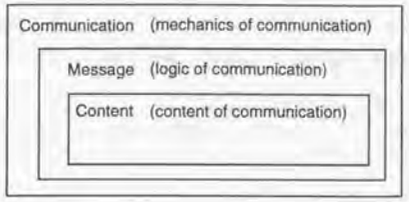



図 2.5: KQML における階層構造 [13]

第 1 章でも述べたように分散人工知能の分野では契約ネットプロトコル [55] を初め多くの協調プロトコルが提案され，分散システムにおけるタスクスケジューリング [44]，トラックの配車問題 [53]，ミーティングスケジューリング [54] など種々の領域に応用されている．また，一方，知識ベースの分野では複数の異種知識ベース間の協調が注目を集めており，知識ベース間の通信言語も提案されている [50]．

### KQML

DARPA の知識共有プロジェクト (Knowledge Sharing Effort)[51] の枠組の中で KQML と呼ばれるエージェント通信言語が提案されている [13]．KQML (Knowledge Query and Manipulation Language) ではエージェント間で交換されるメッセージを次のような三つの層 (layer) から構成されていると考える (図 2.5)．一つは内容層 (content layer) であり，実際のメッセージ内容を記述する．KQML ではどのような表現形式で内容が表現されるかには関与せず，実装上はメッセージ内容の始まりと終りの境界を知っているだけである．また，通信層 (communication layer) ではエージェント間でメッセージを実際に伝達するのに必要な低位の通信パラメータ (例えば，メッセージの宛先，送り手など) に関する記述を行なう．

メッセージ層 (message layer) ではメッセージ行為を記述する．ここが KQML の核となるところで，メッセージ内容に関し遂行語 (performative) を送り手が与えることができる．ここで遂行語はメッセージ内容が例えば問い合わせ (query)，命令 (command) かなどを示す．KQML では多くの遂行語 (performative) を定義しており，発

話行為 (speech act) 論に基づいて意味を与えている [4, 38].

KQML のメッセージではメッセージ内容の記述に用いられている言語や語彙を記述することができる．例えば関西国際空港 (KIX) の緯度，経度を問い合わせする場合のメッセージの一例は次のようになる．

```
(ask-one :content "location(kix,[Longitude,Latitude])"
         :language prolog
         :ontology geo-model3)
```

これは遂行語が ask-one で，prolog 言語で記述された問い合わせが:content 以下に記述されている．また，語彙として geo-model3 が使われていることを示している．ここでの KQML のシンタックスは S 式ベースのものを使用しているが，必ずしもこれに拘る必要はない．

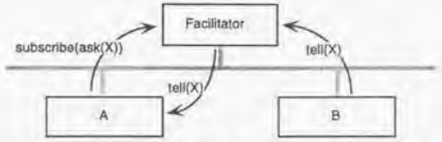
また，KQML では協調促進器 (facilitator) と呼ばれる特殊なエージェントを提案している．協調促進器はエージェント間の仲介の役割を果たすものである．例えば他のエージェントは問い合わせたい事項をあらかじめ協調促進器に通知する (subscribe)．協調促進器がそれ対する答えを得ると subscribe のメッセージを送ったもとのエージェントに対して通知する．その他にもエージェントが自分の能力をあらかじめ協調促進器に通知する (advertise) ことにより，エージェント間の仲介をすることができる．この他にも，broker, recommend などの遂行語が定義されている (図 2.6)．

### COOL

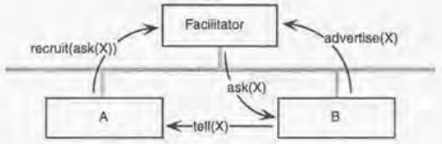
KQML における遂行語を定義しただけではエージェントの意図を表現することはできるものの協調プロトコルの記述としては必ずしも十分でない．そこで，KQML ベースのメッセージに基づいて協調プロトコルを記述するための言語 COOL が提案されている [1]．COOL では状態遷移機械に基づいた会話ルール (conversation rule) でプロトコルを記述するのが特徴になっている．

また，到着したメッセージが現在の状態における会話ルールで処理できない場合に対応するためにエラー回復ルールを定義することができる．エラー回復ルールによって単にメッセージを無視するとか，メッセージについて問い合わせを発するような新たな会話を起動するなどのエラー処理方法を記述することが可能となる．さらに，複

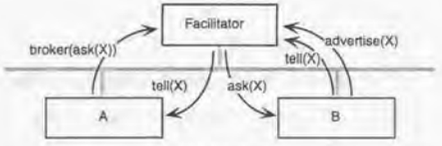




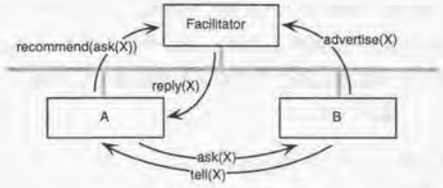

(1) subscribe




(2) recruit




(3) broker




(4) recommend

図 2.6: 協調促進器の例 [13]

数の会話を切替えながら進行できるように継続ルール (continuation rule) を定義できるようになっている.

### CooL

ESPRIT の IMAGINE プロジェクト [14] において提案されたマルチエージェントシステム記述言語 MAI$^2$L[57] から発展した CooL と呼ばれるマルチスレッドの手続き型言語が提案されている [25]. CooL では協調メソッド (cooperation method) として協調プロトコルが記述され, 協調メソッドは領域独立なマルチエージェントプラン (手続き) として表現される. 具体的にはエージェント間のメッセージの交換と個々のエージェントによって実行される手続きから構成される. マルチエージェントプランにおける個々の手続きにはどのエージェントが実行するかが記述されることになる. したがって, COOL が一つのエージェント内の状態遷移を表現しているのに対し, CooL では複数のエージェントの動作を一つにまとめて書くのが特徴になっている.

このような協調プロトコル記述言語を使うことによりマルチエージェントシステムの構築が容易になるものの, 前述の言語では協調プロトコルを段階的に定義することを言語機能としてサポートしていない. マルチエージェントの考え方を使って, 種々多様なアプリケーションを実現しようとするとそれぞれの応用領域に適した協調プロトコルが必要になろう. 特にユーザの代わりとなって働くエージェントの実現では, ユーザ自らプロトコルを記述することも考えられる. このためには, プロトコルを一から記述していくのではなく, 既存のプロトコルをベースにして段階的にプロトコルを定義することにより, 応用に適したプロトコルを容易に実現できることが望まれる.

そこで, 本研究ではプロトコル記述にオブジェクト指向言語に見られるような継承機能を導入し, 段階的なプロトコル定義を可能にした協調プロトコル記述言語 AgenTalk を提案する [31, 32, 33, 34, 37]. 段階的なプロトコル定義機能により, あらかじめ設計したプロトコル群をライブラリーとして保存しておき, 新たなプロトコルを記述する時は既存のプロトコルとの差分だけを定義すればよいという特徴がうまれる.



## 2.5 まとめ

本論文の研究の位置付けを以下にまとめる.

- 協調問題解決においてはエージェント間でどのような情報を交換すべきが大きな課題となっている. 本論文では契約ネットプロトコルの拡張として考えることのできるマルチステージネゴシエーションを取り上げ, そこで対象としている問題の定式化を行ない, 問題の構造に応じてエージェント間で交換すべき情報を規定する. 生の情報すべてを他のエージェントに送るのではなく, いかに情報を抽象化して送るかが問題となる. マルチステージネゴシエーションでは複数のグローバルなゴールが存在するときに大域的な制約を満足させる割当を求めるのが特徴であるが, グローバルゴールを考慮に入れて定式化した探索空間に基づいてエージェントが交換すべき情報を規定することにより, グローバルゴール間の競合が正しく認識できるようにする. さらにここでは, 分散探索という観点から定量的な評価も合わせて行なう.

- 続いて, 市場モデルを分散資源割当に応用し, 資源割当を資源の売買として実現する. マルチステージネゴシエーションでは資源の割当を行なうか行なわないかのいわばゼロ一の問題を扱っているが, これを割り当てる資源の量を扱えるようにする. さらにはエージェントに効用関数を導入し, エージェントは自分の効用を上げるように振舞う. これらは市場モデルを導入することにより, 自然に実現できる. 売り手と買い手というエージェントの種類を考え, エージェント間でやりとりされる情報は資源の価格通知と割当要求に限定する. 従来の研究では価格の決定に際し競売機構の存在を仮定しているが, ここでは売り手のエージェントが競売機構の存在無しに独自に価格を設定するようなメカニズム ("均衡的アプローチ") を提案し, その振舞いを調べる.

- さらに, このようなエージェント間の協調メカニズムを実際に実装するためにはエージェント間のプロトコルを記述するための言語が望まれる. そこで, システムの構築を念頭においた協調プロトコル記述言語 AgenTalk を提案する. AgenTalk では従来のこの種の言語に見られなかったプロトコルの段階的定義機構を導入することを目的とする.

# 第 3 章

# マルチステージネゴシエーションにおけるゴール間競合の検出

## 3.1　はじめに

本章では分散ネットワークにおいて複数のエージェントが資源の割当を協調して行なうマルチステージネゴシエーションにおいて，グローバルなゴール間の競合を検出する手法を提案する．

マルチステージネゴシエーションの概念的な提案は Conry らによってなされている [7]．第 1 章で述べたようにマルチステージネゴシエーションは，通信網において通信路が故障した場合の代替パスの探索を例題として取り上げている．そこでは，通信網は複数の地域に分割され，各々のエージェントが担当地域内の通信路の割当に責任を持つ．通信路が資源割当における資源に対応する．

ここでは，どのエージェントも通信網全体の情報を持たず，各エージェントは他のエージェントに対して自分の局所的な選択 (通信路の割当) の是非を問い合わせ，その問い合わせに対して各エージェントは自分のところにおける影響を返答として返す．文献 [7] では，このようなメッセージの交換を通してエージェントがお互いに満足する資源 (通信路) の割当に到達するという枠組が示されている．そこでは単に枠組が提案されただけで，その前提となる条件，対象としている問題が明確には定義されてはいなかった．また，交渉の過程で実際にどのような形式で情報を交換をすべきかも十分には明らかになっていなかった．

さらに，エージェントの局所的な資源割当の選択がグローバルにどのような影響を

与えるか評価する手法が提案されている [8]．そこではエージェント内における局所的な選択とグローバルなゴールとの競合関係を排除集合 (Exclusion Set) という形で表し，それをエージェント間で交換することによってゴール間の競合の検出を試みている．しかし，そこで提案された手法では一つのゴールを満足するのにグローバルにみて複数の方法（後述するグローバルプランに相当）がありうることを考慮に入れていないため，ゴール間の競合関係が正しく認識できない場合があった [35]．

本章ではまず，文献 [7] で提案されているマルチステージネゴシエーションの概念を基にして，その扱っている問題の定式化を与え，マルチステージネゴシエーションがどういう条件の下で何を解こうとしているかを明確にする．次にその定式化に基づいてどのような情報をエージェントが交換すべきかを明らかにする．そして，交換された情報を用いてどのようにゴール間の競合を検出するかを述べる．

## 3.2 マルチステージネゴシエーションの定式化

### 3.2.1 探索空間

マルチステージネゴシエーションにおいては図 3.1 に示すようにグローバルなゴールがいくつか存在する．グローバルゴールはいわば全体として解くべき目標である．グローバルゴール$g_i$はグローバルプラン$gp_{i,j}$の集合を持つ．グローバルプランとはグローバルゴールを満足するためのエージェントにまたがる仮想的なプランである．グローバルプランはエージェントが局所的に解けるいくつかのサブゴールに分解される．ここで$sg_{i,j,k}$をグローバルプラン$gp_{i,j}$に対する$k$番目のサブゴールとする．それぞれのサブゴールはある唯一のエージェントに属している．一つのサブゴールがいくつかのグローバルプランに同時に属する場合もある．エージェントはそのサブゴールを満足するためのいくつかの方法，すなわちプランフラグメント$pf_{i,j,k,l}$を持つ．プランフラグメントはある唯一のサブゴールに属するものとする．

これらの関係を論理表現を用いて次のように表現することができる．

$$g_i \Leftrightarrow \bigvee_j gp_{i,j} \tag{3.1}$$

$$gp_{i,j} \Leftrightarrow \bigwedge_k sg_{i,j,k} \tag{3.2}$$

$$sg_{i,j,k} \Leftrightarrow \bigvee_l pf_{i,j,k,l} \tag{3.3}$$



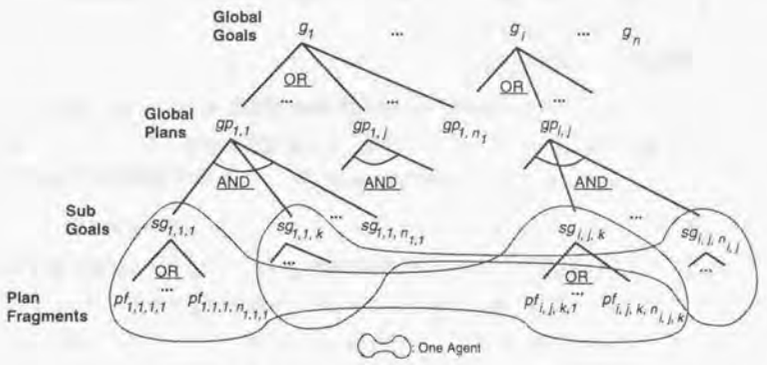



図 3.1: マルチステージネゴシエーションにおける探索空間

ここで$g_i$, $gp_{i,j}$, $sg_{i,j,k}$, $pf_{i,j,k,l}$はそれぞれが満足される，または選択される時に真，そうでない時に偽の値をとるものとする．

### 3.2.2 エージェントの持つ情報

各エージェントは自分のところのサブゴールとそれを満たすプランフラグメントの集合，また，そのサブゴールのグローバルゴールは知っているものの，そのサブゴールがどのグローバルプランに属するかは知らないという状況を考える．これはエージェントの集合全体としてのゴールが定まっていて，各エージェントがそのゴールに対してどのようなことをするべきかがわかっているが，しかし，エージェントの間の連係をどのように取って全体としてうまくゴールを満足できるかがわかっていないという場合と考えることができる．

このためエージェントは自分のサブゴールがどのグローバルプランに属するかを求める必要がでてくる．すなわち，

- エージェントはそのサブゴールとそれを満足するためのプランフラグメントの集合，さらにそのサブゴールが属するグローバルゴールを知っている．
- エージェントはサブゴールがどのグローバルプランに属するかは知らない．こ

のためサブゴールがどのグローバルプランに属するかを求める必要がある．

### 3.2.3 資源割当における制約

ここではプランフラグメントの実行において一定の資源を確保する必要があるとする．資源の割当に関し，エージェント内に閉じた資源使用の制約（エージェント内制約），エージェントにまたがった資源使用の制約（エージェント間制約）の二種類の制約を考える．

- **エージェント内制約**：プランフラグメントを実行するには一定の資源を確保する必要があり，資源の量には限りがあることから同時に選択できるプランフラグメントの組には制限がでてくる．具体的にはプランフラグメントの集合 $PF$ が同時に選択できるためには

$$\sum_{pf_i \in PF} needs(pf_i, r_j) \leq copies(r_j) \ \text{(for all } j) \tag{3.4}$$

が成立する必要がある．ここで $needs(pf_i, r_j)$ は $pf_i$ の実行に必要な資源 $r_j$ の数，また，$copies(r_j)$ は資源 $r_j$ のエージェントにおいて使用可能な数を表す．(簡明に記述するためプランフラグメント，サブゴールの表記を単に添字をつけたものにしている．以下同様)

- **エージェント間制約**：複数のエージェントに関係する資源使用の制約であり，例えばエージェント$A$で資源$ra$が使用される時，$B$では資源$rb$が使用されなければいけないというようなものである．ここで自エージェントとエージェント間制約があるようなエージェントを'関連するエージェント'と呼ぶ．エージェントはすべての自分と関連するエージェントを知っているものとする．

マルチステージネゴシエーションで扱っている問題はこのような資源の割当に制約があり，全体として満足するべきグローバルゴールが複数存在する時に，グローバルゴールを達成するための資源の割当を決めることであるといえる．

### 3.2.4 起動エージェント

グローバルゴール$g_i$は，その$g_i$の解の探索を開始するエージェント（これを起動エージェントと呼ぶ）を一つ持っているとする．起動エージェントは$g_i$を主要ゴール



(*primary goal* – *p-goal*) として扱い，その解の探索の責任を持つ．また，起動エージェントは$g_i$のすべてのグローバルプランに関してそれに属するサブゴールが一つ以上含まれているという仮定をおく．したがって，$g_i$を満足するためには起動エージェントにおいてなんらかのサブゴールが満足されなければならないことになる．

### 3.2.5 問題設定の変更

ここでは問題解決の途中においては問題の設定条件が変化しないという仮定をおく．すなわち，途中でエージェントが追加削除されたり，グローバルゴールが追加削除されたりすることはないとしている．

### 3.2.6 例題

マルチステージネゴシエーションを通信ネットワークに応用した例を示す．この応用ではグローバルゴールはある地点からある地点への通信路を設定することに相当する．エージェントにはそれぞれある地域が割り当てられ，その地域内の通信路の設定に責任を持つ．資源はエージェント地域内の中継点 (Station) または地域間の境界点の間を結ぶ通信路のパスに対応する．また，サブゴールは各エージェント内における地域間の境界点から他の境界点（または，最終目的の通信路の始点ないしは終点）の間を結ぶ通信路の設定に相当する．さらに最終目的となる始点と終点間を結ぶグローバルな通信路がグローバルプランに相当する．これはいくつかのサブゴールが接続されたものになる．

ここでのエージェント内制約とは各通信路の容量の制約に基づく制約である．通信路という性格から地域間の境界点に達する通信路はその境界点の両側でそれぞれ通信路が割り当てられる必要がある．これがエージェント間における資源の割当の制約で，エージェント間制約になる．

図 3.2 に$g1$，$g2$，$g3$の三つのグローバルゴールがある通信ネットワークの例 [26] を示す．ここでは八つのエージェント$A$, $B$, $C$, $D$, $E$, $F$, $G$, $H$がある．それぞれのエージェントは各々の地域の通信路の設定を行なう．$g1$，$g2$，$g3$の起動エージェントはそれぞれ$A$, $B$, $C$である．ここで$g1$とはエージェント$A$から$F$に到達する通信路を求めるもので，また，$g2$はエージェント$B$から$G$に到達する通信路を求めるもので，また，$g3$はエージェント$C$から$H$に到達する通信路を求めるものである．

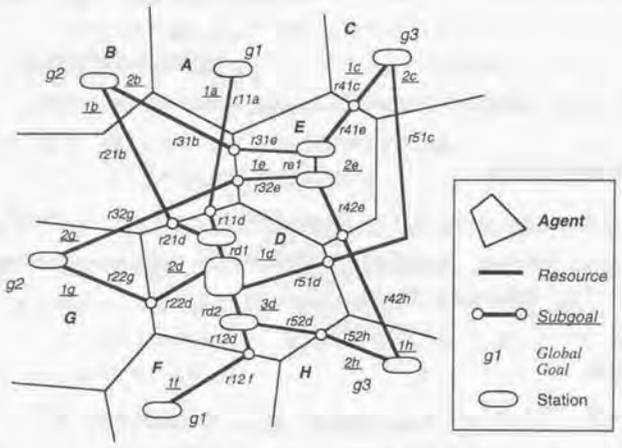



図 3.2: 通信ネットワークの例

グローバルプランとそれを構成するサブゴールを表3.1に示す．この例の場合，$g1$に対して一つ，$g2$，$g3$に対してそれぞれ二つのグローバルプランがある．それぞれのエージェントは表3.2に示すような資源，サブゴールおよびそのプランフラグメントを持っている．表には各プランフラグメントが必要とする資源の数もあわせて示してある．この例では一つのサブゴールに対し一つのプランフラグメントが存在している．

また，この例では次のようなエージェント内制約がある．

- エージェント$D$: プランフラグメント$p1d$と$p2d$はともに資源$rd1$を使用するため同時には選択できない．また，プランフラグメント$p1d$と$p3d$はともに資源$rd2$を使用するため同時には選択できない．
- エージェント$E$: プランフラグメント$p1e$と$p2e$はともに資源$re1$を使用するため同時には選択できない．

ここでのエージェント間制約は同時に使用しなければならない資源の組があるということである．この例では$r11a$と$r11d$，$r21b$と$r21d$，$r31b$と$r31e$，$r41c$と$r41e$，



表 3.1: 例におけるグローバルプラン

| Global Goal | Global Plan | Subgoals |
|---|---|---|
| $g1$ | $gp11$ | $1a$ - $1d$ - $1f$ |
| $g2$ | $gp21$ | $1b$ - $2d$ - $1g$ |
| | $gp22$ | $2b$ - $1e$ - $2g$ |
| $g3$ | $gp31$ | $1c$ - $2e$ - $1h$ |
| | $gp32$ | $2c$ - $3d$ - $2h$ |

$r51c$と$r51d$，$r12d$と$r12f$，$r22d$と$r22g$，$r52d$と$r52h$，$r32e$と$r32g$，$r42e$と$r42h$の資源の組はそれぞれ同時に使用しなければならない．これは，エージェント間の境界につながる通信路については，境界点で通信路が切れてしまわないようにその境界の先のエージェントにおいても資源（通信路）が割り当てられる必要があることに対応する．

## 3.3 競合関係の表現

前節においてマルチステージネゴシエーションが扱っている問題を定義した．交渉を進めていく上でグローバルゴール，グローバルプラン，サブゴール間の競合関係を各エージェントが認識することが必要である．本節では競合関係の表現形式を述べ，さらに次節でどのように競合関係を計算するかを述べる．

### 3.3.1 無効ゴール集合

グローバルゴールが複数存在する時，問題の制約が強過ぎてあるグローバルゴールを満足させると他のグローバルゴールを満足させられないという場合が出てくる．このようなグローバルゴール間の関係を示したものが無効ゴール集合 (Nogood Goal Set) である．$g_i$をグローバルゴール$g_i$が満足された時に真，そうでない時に偽の論理値を持つとして，無効ゴール集合を論理表現を用いて次のように表現する．

$$\bigvee\bigwedge\neg g_i \tag{3.5}$$

表 3.2: 例における各エージェントの資源，サブゴール，プランフラグメント

| Agent $A$ | | | Resource |
|---|---|---|---|
| Goal | Subgoal | pf | $r11a$ |
| Resource Count | | | 1 |
| $g1$ | $1a$ | $p1a$ | 1 |

| Agent $F$ | | | Resource |
|---|---|---|---|
| Goal | Subgoal | pf | $r12f$ |
| Resource Count | | | 1 |
| $g1$ | $1f$ | $p1f$ | 1 |

| Agent $B$ | | | Resource | |
|---|---|---|---|---|
| Goal | Subgoal | pf | $r21b$ | $r31b$ |
| Resource Count | | | 1 | 1 |
| $g2$ | $1b$ | $p1b$ | 1 | |
| | $2b$ | $p2b$ | | 1 |

| Agent $G$ | | | Resource | |
|---|---|---|---|---|
| Goal | Subgoal | pf | $r22g$ | $r32g$ |
| Resource Count | | | 1 | 1 |
| $g2$ | $1g$ | $p1g$ | 1 | |
| | $2g$ | $p2g$ | | 1 |

| Agent $C$ | | | Resource | |
|---|---|---|---|---|
| Goal | Subgoal | pf | $r41c$ | $r51c$ |
| Resource Count | | | 1 | 1 |
| $g3$ | $1c$ | $p1c$ | 1 | |
| | $2c$ | $p2c$ | | 1 |

| Agent $H$ | | | Resource | |
|---|---|---|---|---|
| Goal | Subgoal | pf | $r42h$ | $r52h$ |
| Resource Count | | | 1 | 1 |
| $g3$ | $1h$ | $p1h$ | 1 | |
| | $2h$ | $p2h$ | | 1 |

| Agent $D$ | | | Resource | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Goal | Subgoal | pf | $r11d$ | $r21d$ | $r51d$ | $rd1$ | $rd2$ | $r12d$ | $r22d$ | $r52d$ |
| Resource Count | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $g1$ | $1d$ | $p1d$ | 1 | | | 1 | 1 | 1 | | |
| $g2$ | $2d$ | $p2d$ | | 1 | | 1 | | | 1 | |
| $g3$ | $3d$ | $p3d$ | | | 1 | | 1 | | | 1 |

| Agent $E$ | | | Resource | | | | |
|---|---|---|---|---|---|---|---|
| Goal | Subgoal | pf | $r31e$ | $r41e$ | $re1$ | $r32e$ | $r42e$ |
| Resource Count | | | 1 | 1 | 1 | 1 | 1 |
| $g2$ | $1e$ | $p1e$ | 1 | | 1 | 1 | |
| $g3$ | $2e$ | $p2e$ | | 1 | 1 | | 1 |



例えば

$$\neg g_1 \vee (\neg g_2 \wedge \neg g_3) \tag{3.6}$$

は$g_1$をあきらめるかまたは$g_2$および$g_3$を同時にあきらめる必要があることを示す．最終的には無効ゴール集合が求められればグローバルゴール間の関係がわかり，例えば制約が強過ぎるような場合でもどのグローバルゴールをあきらめればよいかがわかる．もし，同時にすべてのグローバルゴールが満足できるのであれば無効ゴール集合は空となる．

グローバルゴールを満足させるにはそのグローバルプランのうち一つを満足させればいい．したがって，無効ゴール集合を求めるためにはグローバルプランのレベルでの関係を調べる必要がある．また，このようなグローバルプランに関する情報はエージェントの間に分散して存在するのでエージェントの間で情報を交換する必要がある．

### 3.3.2 ゴール排除集合

起動エージェントは自分の主要ゴールに関して他のグローバルゴールのグローバルプランとの相互関係を求め，それを交換しあうことで無効ゴール集合を求めることができる．ここでグローバルゴール$g_i$と競合関係にあるグローバルプラン$gp_{j,k}$ $(j \neq i)$をゴール排除集合 (Goal Exclusion Set) と呼ぶ．この競合関係の意味は次のようになる．

$$\neg g_i \vee (\bigvee \bigwedge_{j \neq i} \neg gp_{j,k}) \tag{3.7}$$

ゴール排除集合は起動エージェントの間で交換され，この情報をもとに起動エージェントは無効ゴール集合を計算する．

### 3.3.3 排除集合

グローバルゴールのゴール排除集合はそのグローバルゴールに含まれるサブゴールの排除集合 (Exclusion Set) から計算される．このサブゴール$sg_{i,j,k}$の排除集合とはそのサブゴールと競合関係にあるグローバルプランの集合である．この競合関係の意味は次のようになる．

$$\neg sg_{i,j,k} \vee (\bigvee \bigwedge_{l \neq i} \neg gp_{l,m}) \tag{3.8}$$

排除集合はサブゴールに対してそれと競合関係にあるグローバルプランを表したものであり，その競合は自分のエージェントの中で発生するものと他のエージェントにおける競合によって起こるものとがある．それを区別して表す時は排除集合をさらに細かく次のように分類する．まず自分の中に閉じた競合だけに基づく排除集合をローカル排除集合 (Local Exclusion Set) と呼ぶ．これに対して他のエージェントにおける競合が直接の原因になるものも含めて考える時は導出排除集合 (Induced Exclusion Set) と呼ぶ．

## 3.4 無効ゴール集合の計算

前節で定義した無効ゴール集合を求める手法を以下説明する．マルチステージネゴシエーションにおいて各エージェントはどのサブゴールを選択したかを関連するエージェントに送信する．ここでどのサブゴールを選択したか通信する際にその選択したサブゴールのグローバルプランに関する情報（後述する選択リスト）を付け加える．エージェントはその情報を基に自分のサブゴールのローカル排除集合を計算する．さらに，自エージェントのサブゴールのローカル排除集合から相手のエージェントの選択したサブゴールの排除集合の情報を計算し，それを相手のエージェントに通知する．

### 3.4.1 競合集合

サブゴールのローカル排除集合を求めるために，そのサブゴールとは同時には選択できないサブゴールの集合（これを競合集合 (Conflict Set) と呼ぶ）を考える．まず，その準備としてプランフラグメントの集合 $P_k = \{pf_i\}$ は以下の条件が成り立つとき，“互換性がある (compatible)” と定義する[1]．

$$\sum_{pf_i \in P_k} needs(pf_i, r_j) \leq copies(r_j) \ \text{(for all } j) \tag{3.9}$$

ここで $needs(pf_i, r_j)$ はプランフラグメント $pf_i$ の実行に必要な資源 $r_j$ の数，$copies(r_j)$ はこのエージェントによって使用可能な資源 $r_j$ の数を表す．

[1]文献 [8] ではサブゴールの考え方がないため，こことは違った意味で “互換性がある (compatible)” という言葉を用いている．



また，$SG_A$ をエージェント $A$ のサブゴールの集合とする．サブゴール $sg_i$ のグローバルゴールを $goal(sg_i)$ で表す．さらにサブゴール $sg_i$ のプランフラグメントの集合を $P(sg_i)$ とする．

ここで $SG_A$ の部分集合で，その要素のサブゴールのグローバルゴールがすべて相異なる集合 $SSG_k$ を考える．すなわち，

$$SSG_k = \{sg_i \mid goal(sg_i) \neq goal(sg_j) \text{ for all } i, j\} \tag{3.10}$$

ここで $_k$ はこのような集合が一つのエージェントに複数存在することを示すためにつけた添字である．$SSG_k$ が互換性があるとは $SSG_k$ のサブゴールのすべてを満たすような互換性があるプランフラグメントの集合が存在することと定義する．すなわち，

$$\begin{aligned} &SSG_k\text{が互換性がある} \Leftrightarrow \\ &\quad \exists P_l\,[\,P_l\text{が互換性がある} \wedge \forall sg_i \in SSG_k\,[\,(P(sg_i) \cap P_l) \neq \phi\,]\,] \end{aligned} \tag{3.11}$$

論理表現を用いると $SSG_k = \{sg_1, sg_2, ..., sg_{n_k}\}$ が互換性がないということは次のように表せる．

$$\neg(\bigwedge_{sg_i \in SSG_k} sg_i) \tag{3.12}$$

ここで，サブゴール $sg_i$ を含み，かつ，互換性がないすべての $SSG_k$ の集合 $SS_{sg_i}$ を考える．すなわち，

$$SS_{sg_i} = \{SSG_k \mid sg_i \in SSG_k \wedge SSG_k \text{ が互換性がない}\} \tag{3.13}$$

この集合の意味は

$$\bigwedge_{SSG_k \in SS_{sg_i}} \neg(\bigwedge_{sg_j \in SSG_k} sg_j) \tag{3.14}$$

である．$SSG_k \in SS_{sg_i}$ は $sg_i$ を含んでいるのでこの式は次のように変形できる．

$$\neg sg_i \vee (\bigwedge_{SSG_k \in SS_{sg_i}} \bigvee_{sg_j \in SSG_k, j \neq i} (\neg sg_j)) \tag{3.15}$$

$sg_i$ の競合集合 $CS_{sg_i}$ は $(\bigwedge_{SSG_k \in SS_{sg_i}} \bigvee_{sg_j \in SSG_k, j \neq i} (\neg sg_j))$ をサブゴールの否定の選言標準形に変換した形で表現する．

### 3.4.2 グローバルプランの表現

競合集合からローカル排除集合を計算するためにはサブゴールがどのグローバルプランに属しているかを記述できるようにする必要がある．ここではサブゴールの性質として選択リストというものを導入し，グローバルゴールと選択リストの組でサブゴールの属するグローバルプラン（場合によっては複数の場合もある）を表現する．サブゴール$sg_i$の選択リストを$cl(sg_i)$と表し，$sg_i$の属するグローバルプランを次のように書く．

$$<goal(sg_i), cl(sg_i)> \tag{3.16}$$

この記述をここではゴール記述と呼ぶ．

エージェントはどのようなグローバルプランがあるかはあらかじめ知らず，選択リストを交渉の過程で段階的に作成することによってどのようなグローバルプランがあるかを認識できるようになる．

#### 選択リストの構成要素

選択リストは各エージェントにおけるいくつかあるサブゴールのうちどのサブゴールを選択したかを表したもので，具体的には*local-id*の選言標準形で表される．*local-id*とはそれぞれのエージェントでどのサブゴールを選んだかという選択を表すものである．ここでは起動エージェントにおける選択と起動エージェントでないエージェントにおける選択とを区別して考える．

起動エージェントにおける選択を表す*local-id*は次のような形式で表される．

$$(\ agent\ index\ total\text{-}number\ ) \tag{3.17}$$

ここで*agent*は起動エージェント，*index* ($1 \leq index \leq total\text{-}number$) はその選択が全部で何通りかある選択枝のうち何番目のものかを表し，そして*total-number*は選択肢の全体の数を表す．*total-number*が含まれているのはどのエージェントにおいてもすべての選択肢と競合しているかを容易に識別できるようにするためである．もし，エージェントの選択肢が一つしかない場合は*index*と*total-number*は省略され，単に

$$(\ agent\ ) \tag{3.18}$$

と書かれる．



一方，起動エージェント以外のエージェントにおける選択を表す*local-id*は次のような形式で表される．

$$(\ entry\text{-}id\ index\ total\text{-}number\ ) \tag{3.19}$$

ここで*entry-id*はエージェントの内部テーブルのエントリを示すIDである．この内部テーブルとは他のエージェントから送られてくる選択の確認のメッセージに対して，自エージェント内で選択するサブゴールの候補を得るのに用いられるものである．このメッセージには選択されたサブゴールとサブゴールを選択した時に割り当てられた資源のうちエージェント間制約に関係する資源が含まれている．内部テーブルの各エントリはグローバルゴール（メッセージ中のサブゴールから求められる）とエージェント間制約の資源をキーとしてそれに対する自エージェント内で選択可能なサブゴールのリストを与える．*index*はサブゴールのリストの中で選択されたサブゴールが何番目のものかを示す．また，*total-number*はその候補のリストの要素の数を示す．ただし，候補が一つしかない場合は*local-id*そのものが省略される．

#### 選択リストの生成

まず，起動エージェントは自分の主要ゴール (p-goal) を調べ，どのサブゴールを用いるかを決め，それを関連するエージェントに送信する．この時の選択を表した*local-id*がそのまま選択されたサブゴールの選択リストとなる．

起動エージェント以外のエージェントにおいては他のエージェントからその仮の選択の確認のメッセージを受け取った時はまず，その内部テーブルを参照して自分の中のどのサブゴールを選択するか決める．この時メッセージに含まれているサブゴールを$sg_o$，自エージェント内での選択を表す*local-id*を$lid_h$，自エージェント内で選択されたサブゴールを$sg_h$とする．この時，$sg_h$の選択リスト$cl(sg_h)$は次のように更新される．

$$cl(sg_h) = cl(sg_h) \vee (cl(sg_o) \wedge lid_h) \tag{3.20}$$

ここで自エージェント内での候補が一つしかく*local-id*が省略された時は$lid_h$は真として計算する．また，サブゴールの選択リストの初期値は偽の値を持つものとして計算する．

形式的には選択リスト (choice list) は次のようになる．

$$\textit{choice list} \;=\; \bigvee\bigwedge \textit{local-id} \tag{3.21}$$

$$\textit{local-id} \;=\; (\textit{agent}\ [\textit{index total-number}\,]) \quad \text{または}\ (\ \textit{entry-id}\ [\textit{index total-number}\,]) \tag{3.22}$$

となる．ここで [, ] は省略される場合があることを表す．

**選択リストの簡単化ルール**

選択リストは *local-id*の選言標準形で表現され，さらに同じ選択に関してすべてのインデックスが集められた時，それらは次の簡単化ルールにしたがってまとめられる．

$$\bigvee_{index=1,\ldots,total-number} (\ \{\textit{agent} \mid \textit{entry-id}\}\ \textit{index total-number}\ ) \Rightarrow (\ \{\textit{agent} \mid \textit{entry-id}\}) \tag{3.23}$$

ただし，ここで $\{agent \mid entry\text{-}id\}$ は *agent*か *entry-id*のどちらかであることを示す．*local-id*は *total-number*を含んでいるのでこの簡単化はどのエージェントにおいても行なうことができる．

ここで選択リストを選言標準形に変換した時そのすべての連言要素が'(*agent*)'の形式の表現を含む時，その選択リストを'主要ゴールを含む'と呼ぶ．これは起動エージェントにおけるすべての選択を含むことを意味する．問題設定の仮定から起動エージェントにはすべてのグローバルゴールに関してそれに属するサブゴールが一つ以上含まれているので，起動エージェントにおけるすべての選択とはすべてのグローバルプランの選択を意味していることになる．言い変えれば

$$\text{'主要ゴールを含む'選択リスト}\ (\textit{choice list}) = \bigvee(\ (\ \textit{agent}\ )\ [\wedge\ (\ \bigwedge\ \textit{local-id}\ )\ ]) \tag{3.24}$$

のように表せる．

**選択リストの計算例**



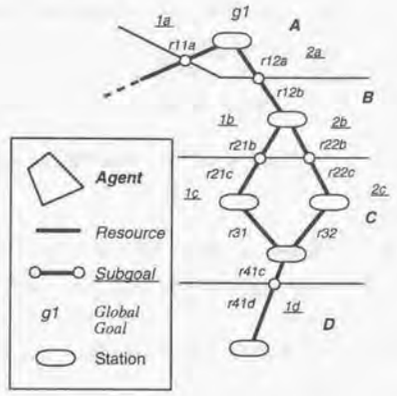



図 3.3: 通信ネットワークの別例

図 3.3 のような通信路の一部では選択リストは次のようになる．起動エージェント $A$にサブゴール1*a*と2*a*がある．この時，2*a*の選択リストは

$$cl(2a) \;=\; (A\ 2\ 2) \tag{3.25}$$

となる．エージェント $B$にはそのサブゴールと同じグローバルプラン (この場合二つある) に属するサブゴール1*b*と2*b*がある．これらのサブゴールの選択リストは

$$cl(1b) \;=\; (A\ 2\ 2)\wedge(\mathrm{B1}\ 1\ 2) \tag{3.26}$$

$$cl(2b) \;=\; (A\ 2\ 2)\wedge(\mathrm{B1}\ 2\ 2) \tag{3.27}$$

となる．ただし，B1 はエージェント $B$において2*a*と接続する場合の *entry-id*である．さらにエージェント $C$にはエージェント $B$のサブゴール1*b*, 2*b*とそれぞれ同じグローバルプランに属するサブゴール1*c*, 2*c*がある．ここではエージェント $C$において $B$でサブゴールが決定してしまえば選択の余地が残されていないので選択リストは変化しない．すなわち，

$$cl(1c) \;=\; cl(1b) \tag{3.28}$$

$$cl(2c) \;=\; cl(2b) \tag{3.29}$$

となる．ここでまずエージェント$C$でサブゴール1$c$が選択されたとするとエージェント$D$ではサブゴール1$d$が選択され，1$d$の選択リストは

$$cl(1d) = cl(1b) \tag{3.30}$$

となる．さらにエージェント$C$で新たにサブゴール2$c$が選択されたとするとサブゴール1$d$の選択リストは

$$cl(1d) = cl(1b) \vee cl(2b) \tag{3.31}$$
$$= ((A\ 2\ 2) \wedge (\mathrm{B1}\ 1\ 2)) \vee ((A\ 2\ 2) \wedge (\mathrm{B1}\ 2\ 2)) \tag{3.32}$$

となり，(3.23) の簡単化ルールを適用すると

$$cl(1d) = (A\ 2\ 2) \wedge (\mathrm{B1}) \tag{3.33}$$

となる．ここで1$d$は二つのグローバルプランに属しており，この例からもわかるように選択リストはそのような場合に対してもコンパクトな表現形式を与える．

### 3.4.3　排除集合の結合

前節で述べたようにサブゴールの属するグローバルプランはゴール記述を用いて表現される．排除集合はサブゴールと競合関係にあるグローバルプランを表したものであり，具体的にはゴール記述の否定の選言標準形で表現する．

このとき排除集合において同じグローバルゴールに関してはゴール記述は次のようなマージルールにしたがってまとめる．

$$\bigvee\bigwedge \neg <g_i, cl(sg_j)> \Rightarrow \neg <g_i, \bigwedge \vee cl(sg_j)> \tag{3.34}$$

ここでゴール記述の選択リストが'主要ゴールを含む'であれば，そのゴール記述も同様に'主要ゴールを含む'と言う．もし，ゴール記述が'主要ゴールを含む'であればそれはグローバルゴールと意味的には等価である．

#### ローカル排除集合

ローカル排除集合はエージェント内に閉じた競合を表したものである．サブゴール$sg_i$のローカル排除集合 $LES_{sg_i}$ は$sg_i$の競合集合 $CS_{sg_i}$ に現れるサブゴールをその



ゴール記述で置き換えたものである．すなわち，$CS_{sg_i} = \bigvee\bigwedge \neg sg_j$ である時，

$$LES_{sg_i} = \bigvee\bigwedge \neg <goal(sg_j), cl(sg_j)> \tag{3.35}$$

となる．

### 3.4.4　導出排除集合

サブゴール$sg_i$に対する導出排除集合 $IES_{sg_i}$ は$sg_i$を選択した時の局所的かつ非局所的な影響を表しているものと考えることができる．導出排除集合は最初はローカル排除集合と同じ値に設定され，他のエージェントから情報を受け取るにしたがい変化していく．この時，どのような排除集合の情報を他のエージェントに送ればよいか，そして，他のエージェントから送られてきた排除集合の情報をどのように結合すればよいかが問題となる．

#### 送信する排除集合の情報の計算

エージェント$X$のサブゴール$sg_x$に対して，エージェント$A$では$sg_x$と同じグローバルプランに属する$n$個のサブゴール$sg_1, \ldots, sg_n$があるとする．これは$sg_x$が複数のグローバルプランに属し，エージェント$A$ではそれらのグローバルプランが$n$個のサブゴールに分かれている場合に相当する．$X$が$sg_x$の導出排除集合を計算するためには$A$は$sg_1, \ldots, sg_n$の導出排除集合を結合し，$X$に通信する必要がある．

エージェント$A$では次の式が成り立つ．

$$\neg sg_i \vee IES_{sg_i}\ (1 \leq i \leq n) \tag{3.36}$$

もし，$sg_x$が最終的な解として選択されるならば$sg_i$のどれかが選択されることになる．逆にもし$sg_i$のどれかが最終解で選択されるならば，$sg_x$も選択されることになる．したがって，次の式が成り立つ．

$$sg_x \Leftrightarrow \bigvee_i sg_i \tag{3.37}$$

(3.37) から

$$\neg sg_x \vee (\bigvee_i sg_i) \tag{3.38}$$

(3.36) と (3.38) から次の式が導ける．

$$\neg sg_x \vee (\bigvee_i IES_{sg_i}) \tag{3.39}$$

したがって，エージェント$A$は次の排除集合の情報$ES_A$をエージェント$X$に送ればいい．

$$ES_A = \bigvee_i IES_{sg_i} \tag{3.40}$$

これはエージェント$X$からみると$A$に仮想的な一つのサブゴール$sg_A$が存在し，

$$sg_A \Leftrightarrow sg_x \tag{3.41}$$

$$\neg sg_A \vee ES_A \tag{3.42}$$

の関係にあると考えることができる．

**受け取った排除集合の情報の合成**

エージェント$A$においてサブゴール$sg_a$の導出排除集合を計算するのにエージェント$X_i(1 \leq i \leq n)$とエージェント間制約があり，これらエージェントから排除集合の情報$ES_{X_i}$をもらう必要があるとする．この時これらの情報をまとめ，$sg_a$の導出排除集合を更新する必要がある．

それぞれのメッセージに含まれる排除集合の情報$ES_{X_i}$はエージェント$X_i$の仮想的なサブゴール$sg_{X_i}$と次のような関係がある．

$$\neg sg_{X_i} \vee ES_{X_i} \tag{3.43}$$

ここで$sg_a$が選択された時，すべての$i$に対し$sg_{X_i}$も同時に選択される．逆にすべての$i$に対し$sg_{X_i}$が最終的に選択されるならば$sg_a$も同時に選択されることになる．したがって，次の式が成り立つ．

$$sg_a \Leftrightarrow \bigwedge_i sg_{X_i} \tag{3.44}$$

(3.44) より，

$$\neg sg_a \vee (\bigwedge_i sg_{X_i}) \tag{3.45}$$

(3.43) と (3.45) から，

$$\neg sg_a \vee (\bigwedge_i ES_{X_i}) \tag{3.46}$$



が導ける．したがって$sg_a$の導出排除集合$IES_{sg_a}$は次のようになる．

$$IES_{sg_a} = (\bigwedge_i ES_{X_i}) \wedge LES_{sg_a} \tag{3.47}$$

### 3.4.5 ゴール排除集合

ここでグローバルゴール$g_i$がその起動エージェントに$n$個のサブゴール$sg_1$, ..., $sg_n$を持っているとする．これらの関係は次のように表現できる．

$$g_i \Rightarrow \bigvee_i sg_i \tag{3.48}$$

ここで$IES_{sg_i}$を$sg_i$の導出排除集合とする．すなわち，

$$\neg sg_i \vee IES_{sg_i} \tag{3.49}$$

(3.48) から，

$$\neg g_i \vee (\bigvee_i sg_i) \tag{3.50}$$

(3.49) と (3.50) から，

$$\neg g_i \vee (\bigvee_i IES_{sg_i}) \tag{3.51}$$

したがって，グローバルゴール$g_i$のゴール排除集合$GES_{g_i}$は次のように計算される．

$$GES_{g_i} = \bigvee_i IES_{sg_i} \tag{3.52}$$

$GES_{g_i}$は$g_i$を満足させることが与える他のグローバルゴールのグローバルプランへの影響，すなわち，$g_i$との競合関係にあるゴール記述を示しているといえる．

### 3.4.6 無効ゴール集合

起動エージェントはまず自分のグローバルゴールに関しゴール排除集合を計算し，それを他の関連する起動エージェントに送る．送られてきたゴール排除集合は起動エージェントの中で導出ゴール排除集合に追加されていく．この導出ゴール排除集合とは起動エージェントに一つできるもので，他の起動エージェントから送られるゴール排除集合を蓄えるものである．ここでゴール排除集合を送る相手の起動エージェントとはこの導出ゴール排除集合の中に出てくるグローバルゴールを‘主要ゴール’とし

て持つ起動エージェントのことである．なお自エージェント内で求められたゴール排除集合もそのエージェント内の導出ゴール排除集合に追加される．

起動エージェント$A$に対する導出ゴール排除集合$IGES_A$は次のように計算される．

$$IGES_A = (\bigwedge_{g_i \in IG_A} (\neg g_i \vee GES_{g_i})) \wedge (\bigwedge_{g_j \in PG_A} (\neg g_j \vee GES_{g_j})) \quad (3.53)$$

ここで$IG_A$は起動エージェント$A$の主要ゴールと直接，間接的に関係するグローバルゴール，すなわち，$IGES_A$に出現する自分の主要ゴール以外のすべてのグローバルゴールの集合を示す．また，$PG_A$はエージェント$A$の主要ゴールの集合を示す．

各起動エージェントは，この導出ゴール排除集合に次のような操作を施すことにより無効ゴール集合を計算する．

1. 導出ゴール排除集合を選言標準形

$$F_1 \vee F_2 \vee \ldots \vee F_n \quad (3.54)$$

に直す．ただし，

$$F_i = (\bigwedge_j \neg g_j) \wedge (\bigwedge_k GES_{g_k}) \quad (3.55)$$

である．

2. 選言標準形中の各連言$F_i$に次の操作を施す．

   (a) 同じグローバルゴールに関して(3.34)のマージルールにしたがいまとめる．

   (b) 各選択リストに関して(3.23)の簡単化ルールにしたがって簡単化する．

   (c) '主要ゴールを含む' ゴール記述をグローバルゴールに置き換える．

   (d) もし，各連言$F_i$中に一つでもグローバルゴールが含まれれば$F_i$の中からグローバルゴールでないものを取り除く．これは一般に命題$a$,$b$,$c$の間には次の式が成り立つことによる．

$$(a \wedge b) \vee c \Rightarrow a \vee c \quad (3.56)$$



3. ここでグローバルゴール以外のものが残ったならば無効ゴール集合は空とする．そうでなければ書き換えられた各連言$F_i'$に関して

$$F_i' \Rightarrow F_j' \quad (i \neq j) \quad (3.57)$$

となる$F_i'$をすべて取り除く．これは無効ゴール集合に関して最小のもの，すなわち，満足されないグローバルゴールが最小になる場合を考えるためである．

### 3.4.7 例題

第2節の通信路の例題における無効ゴール集合を計算してみる．この場合の各エージェントにおける選択リスト，排除集合は最終的には表3.3のようになる．そして，各起動エージェントにおけるゴール排除集合は表3.4のようになる．このゴール排除集合は他の起動エージェントに送られて，結局すべての起動エージェントの導出ゴール排除集合$IGES$は

$$\begin{aligned} IGES &= (\neg g1 \vee (\neg{<}g2, (B\ 1\ 2){>} \wedge \neg{<}g3, (C\ 2\ 2){>})) \\ &\quad \wedge (\neg g2 \vee \neg{<}g1, (A){>} \vee \neg{<}g3, (C\ 1\ 2){>}) \\ &\quad \wedge (\neg g3 \vee \neg{<}g2, (B\ 2\ 2){>} \vee \neg{<}g1, (A){>}) \quad (3.58) \\ &= (\neg g1 \vee {<}g2, (B\ 1\ 2){>}) \\ &\quad \wedge (\neg g1 \vee {<}g3, (C\ 2\ 2){>}) \\ &\quad \wedge (\neg g2 \vee \neg{<}g1, (A){>} \vee \neg{<}g3, (C\ 1\ 2){>}) \\ &\quad \wedge (\neg g3 \vee \neg{<}g2, (B\ 2\ 2){>} \vee \neg{<}g1, (A){>}) \quad (3.59) \end{aligned}$$

となる．各起動エージェントはこれを次のように変形する．

選言標準形に変換(ステップ1)

$$\begin{aligned} IGES &= (\neg g1 \wedge \neg g2 \wedge \neg g3) \vee \ldots \\ &\quad \vee (\neg g1 \wedge \neg{<}g1, (A){>}) \vee \ldots \\ &\quad \vee (\neg{<}g2, (B\ 1\ 2){>} \wedge \neg{<}g3, (C\ 2\ 2){>} \\ &\qquad \wedge \neg g2 \wedge \neg{<}g2, (B\ 2\ 2){>}) \vee \ldots \\ &\quad \vee (\neg{<}g2, (B\ 1\ 2){>} \wedge \neg{<}g3, (C\ 2\ 2){>} \end{aligned}$$

表 3.3: 例における選択リスト (Choice List)，排除集合 (Exclusion Set)

| Agent | Global Goal | Sub-goal | Choice List | Local Exclusion Set | Induced Exclusion Set |
|---|---|---|---|---|---|
| $A$ | $g1$ | $1a$ | $(A)$ | | $\neg<g2,(B\ 1\ 2)> \wedge \neg<g3,(C\ 2\ 2)>$ |
| $B$ | $g2$ | $1b$ | $(B\ 1\ 2)$ | | $\neg<g1,(A)>$ |
| | $g2$ | $2b$ | $(B\ 2\ 2)$ | | $\neg<g3,(C\ 1\ 2)>$ |
| $C$ | $g3$ | $1c$ | $(C\ 1\ 2)$ | | $\neg<g2,(B\ 2\ 2)>$ |
| | $g3$ | $2c$ | $(C\ 2\ 2)$ | | $\neg<g1,(A)>$ |
| $D$ | $g1$ | $1d$ | $(A)$ | $\neg<g2,(B\ 1\ 2)> \wedge \neg<g3,(C\ 2\ 2)>$ | |
| | $g2$ | $2d$ | $(B\ 1\ 2)$ | $\neg<g1,(A)>$ | |
| | $g3$ | $3d$ | $(C\ 2\ 2)$ | $\neg<g1,(A)>$ | |
| $E$ | $g2$ | $1e$ | $(B\ 2\ 2)$ | $\neg<g3,(C\ 1\ 2)>$ | |
| | $g3$ | $2e$ | $(C\ 1\ 2)$ | $\neg<g2,(B\ 2\ 2)>$ | |
| $F$ | $g1$ | $1f$ | $(A)$ | | |
| $G$ | $g2$ | $1g$ | $(B\ 1\ 2)$ | | |
| | $g2$ | $2g$ | $(B\ 2\ 2)$ | | |
| $H$ | $g3$ | $1h$ | $(C\ 1\ 2)$ | | |
| | $g3$ | $2h$ | $(C\ 2\ 2)$ | | |

表 3.4: 例におけるゴール排除集合 (Goal Exclusion Set)

| Agent | p-goal | Goal Exclusion Set |
|---|---|---|
| $A$ | $g1$ | $\neg<g2,(B\ 1\ 2)> \wedge \neg<g3,(C\ 2\ 2)>$ |
| $B$ | $g2$ | $\neg<g1,(A)> \vee \neg<g3,(C\ 1\ 2)>$ |
| $C$ | $g3$ | $\neg<g2,(B\ 2\ 2)> \vee \neg<g1,(A)>$ |



$$\wedge\neg<g3,(C\ 1\ 2)> \wedge \neg g3) \vee \ldots \quad (3.60)$$

グローバルゴールに関してマージ (ステップ 2a)

$$\Rightarrow\ (\neg g1 \wedge \neg g2 \wedge \neg g3) \vee \ldots$$
$$\vee(\neg g1 \wedge \neg<g1,(A)>) \vee \ldots$$
$$\vee(\neg<g2,(B\ 1\ 2)\wedge(B\ 2\ 2)> \wedge \neg<g3,(C\ 2\ 2)> \wedge \neg g2) \vee \ldots$$
$$\vee(\neg<g2,(B\ 1\ 2)> \wedge \neg<g3,(C\ 1\ 2)\wedge(C\ 2\ 2)> \wedge \neg g3) \vee \ldots \quad (3.61)$$

選択リストの簡単化 (ステップ 2b)

$$\Rightarrow\ (\neg g1 \wedge \neg g2 \wedge \neg g3) \vee \ldots$$
$$\vee(\neg g1 \wedge \neg<g1,(A)>) \vee \ldots$$
$$\vee(\neg<g2,(B)> \wedge \neg<g3,(C\ 2\ 2)> \wedge \neg g2) \vee \ldots$$
$$\vee(\neg<g2,(B\ 1\ 2)> \wedge \neg<g3,(C)> \wedge \neg g3) \vee \ldots \quad (3.62)$$

グローバルゴールへの置き換え (ステップ 2c)

$$\Rightarrow\ (\neg g1 \wedge \neg g2 \wedge \neg g3) \vee \ldots \vee \neg g1 \vee \ldots$$
$$\vee(\neg g2 \wedge \neg<g3,(C\ 2\ 2)>) \vee \ldots$$
$$\vee(\neg<g2,(B\ 1\ 2)> \wedge \neg g3) \vee \ldots \quad (3.63)$$

グローバルゴール以外のものの除去 (ステップ 2d)

$$\Rightarrow\ (\neg g1 \wedge \neg g2 \wedge \neg g3) \vee \ldots$$
$$\vee\neg g1 \vee \ldots \vee \neg g2 \vee \ldots \vee \neg g3 \vee \ldots \quad (3.64)$$

最小化 (ステップ 3)

$$\Rightarrow\ \neg g1 \vee \neg g2 \vee \neg g3 \quad (3.65)$$

このことはこの例題では制約が強過ぎてすべてのグローバルゴールを同時には満たすことができなく，どれか一つのグローバルゴールをあきらめることが必要なことを示している．

## 3.5 議論

ここで提案した手法ではエージェントのすべての情報を一個所に集中する場合に比べ，起動エージェントに排除集合という形式で抽象化された情報が集まるので他の

エージェントの内部に関して知る必要がないという利点がある．また，逆にエージェントはその内部情報を洗いざらい他のエージェントに通信する必要もない．起動エージェントがゴール排除集合を計算するため起動エージェントに対する負荷が大きくなるおそれがあるものの，グローバルな競合関係を認識するためには，ここで述べたような情報を集める必要がある．

また，各エージェントにおいてはその排除集合の内容から自分の局所的な選択の影響を認識することができ，無駄な試みを減らすことが可能になる．例えば無効ゴール集合からどのグローバルゴールを同時に満足すべきかを決定した後，どのグローバルプランを用いてそのグローバルゴールを満足させるか決定するときゴール排除集合からグローバルゴールと競合関係にあるグローバルプランが何かわかるので，その情報より選択すべきグローバルプランを決めることが可能になる．さらにエージェント内でどのサブゴールを選択するか決める場合，サブゴールの導出排除集合からどのグローバルプランと競合しているかわかるので，競合関係にあるサブゴールを避けることができる．

また，制約が強過ぎる場合は無効ゴール集合からどのグローバルゴールを解いたらよいかを決めることができる．ここでは無効ゴール集合が求まった後はどのグローバルゴールを解くべきかは起動エージェントの間で合意されている何らかの基準で一意に決まるものと仮定している．例えば解くべきグローバルゴールの数を最大にする，または，グローバルゴールにエージェント間で合意のとれた効用があらかじめ決まっているとして，効用の総和が最大になるように決めるなどである．例題では無効ゴール集合が $\neg g1 \vee \neg g2 \vee \neg g3$ であり，仮に$g1$，$g2$，$g3$の効用がそれぞれ 10，20，30 だとすると，$g1$をあきらめることによって$g2$と$g3$が満足でき，解くグローバルゴールの効用の和を 50 と最大にすることができる．したがって，$g1$をあきらめるべきと決めることができる．もし，このようなエージェント間で合意のとれた基準がない場合は，どのグローバルゴールを解くかについて無効ゴール集合の情報に基づいて起動エージェントの間で別の交渉を行なう必要がでてくる．

## 3.6 まとめ

本章ではマルチステージネゴシエーションを定式化し，エージェント間で交換すべき情報の形式を明らかにした．この定式化ではグローバルゴールを満足させるための



（仮想的な）グローバルプランを考え，各グローバルプランは各エージェントにおいてサブゴールに分解される．エージェント内のサブゴールは各エージェント内のプランフラグメントを実行することによって満足されるとした．

ここでは，ゴール記述というものを取り入れグローバルプランをコンパクトに表現し，グローバルプランの情報をやりとりすることにより，グローバルゴールの競合を正しく認識できるようにした．

# 第 4 章

# マルチステージネゴシエーションにおける探索戦略の評価

## 4.1 はじめに

前章で述べたようにマルチステージネゴシエーションではエージェント間で資源の割当及びその影響に関する情報の交換を行なう必要があり，どのような情報を交換したらよいかがマルチステージネゴシエーションにおける課題の一つであった [36]．さらにエージェント間で交換した情報に基づいてエージェントがどのように振舞うべきかも問題となる．マルチステージネゴシエーションではその探索戦略として 3 フェーズプロトコルを提案している [6, 26, 35]．このプロトコルはマルチステージネゴシエーションで定義されているグローバルゴール，グローバルプランなどの概念に基づいている．さらにこのプロトコルを一種の探索戦略と見なした場合どのような性質を持つか，定量的評価を行なう．ここではマルチステージネゴシエーションで扱っている問題をより一般的な枠組でとらえなおし，その枠組で用いられている汎用のアルゴリズムと性能を比較することにより 3 フェーズプロトコルの性質を明らかにする．

マルチステージネゴシエーションで扱っている問題は制約充足問題 [43] が複数のエージェントに分散している分散制約充足問題と考えることもできる [63, 64]．そこでマルチステージネゴシエーションで扱っている問題を分散制約充足問題へマッピングする．同一の問題をマルチステージネゴシエーションおよび分散制約充足問題としてとらえ，各々の手法で解いてみることにより，3 フェーズプロトコルの定量的な性質を比較・評価する．ここでは次のような評価方針を立てた．

- 分散制約充足問題ではいくつかの解法 [5, 63, 64] が提案されているが，その中の基本的な解法の一つである非同期バックトラック [63, 64] との比較を行なう．このアルゴルリズムを選んだのはこのアルゴリズムをベースに性能評価例が報告されており [48]，分散制約充足問題において比較の基盤を与える基本的なアルゴリズムの一つであると考えられるからである．

- 実際の応用に近い問題で評価するため NTT の通信網のデータより作成した通信網のパスの設定問題を例題とする．

まず，マルチステージネゴシエーションとその探索戦略の概略を述べ，次に分散制約充足問題とその解法である非同期バックトラックの概略を述べる．さらに，マルチステージネゴシエーションで扱われている問題が分散制約充足問題にどのようにマッピングされるかを述べる．そして，性能比較の実験の結果を基にマルチステージネゴシエーションと非同期バックトラックの両者の性能を比較し，その得失を議論する．

## 4.2 3 フェーズプロトコル

マルチステージネゴシエーションではエージェントはあるサブゴールを選択する場合そのサブゴールを満足するのに必要なプランフラグメントが必要としている資源を調べ，他のエージェントとエージェント間制約となっているものがあればそのエージェントに対し，その資源を使用して問題がないかの確認のメッセージを送る．メッセージを受け取ったエージェントはエージェント間制約を満足するようにサブゴールを選択できるか調べる．ここでもし可能ならば OK を返答し，だめならばその競合情報を通知する．このようなメッセージのやりとりを一つのステージとしてこれを何回も繰り返すところがマルチステージネゴシエーションの基本的な考え方である．しかしそれだけではグローバルゴールを満足させる解があるにもかかわらず見落としてしまう危険がある．そこでマルチステージネゴシエーションにおける探索戦略として非同期探索，協調探索，過制約解消の三つのフェーズからなる 3 フェーズプロトコルを導入している (図 4.1)．

ここでの基本的な考え方はまず複数のグローバルゴールに関してばらばらに解を求めようとして，すべての解が求まればその時点で終了する．もし，求まらなければ前述のエージェント間の競合関係をすべて計算するというものである．さらにここで



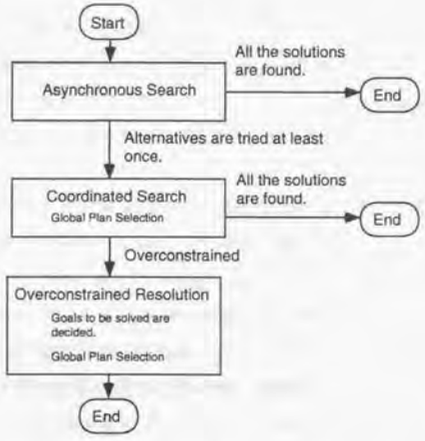



図 4.1: 3 フェーズプロトコル

は，無効ゴール集合が空でなく，制約が強過ぎることがわかったならば適当なグローバルゴールをあきらめて残りのグローバルゴールのみを満足させるという方針をとっている．

#### 4.2.1 非同期探索フェーズ

非同期探索フェーズ (Asynchronous Search Phase) では起動エージェントはその主要ゴール (p-goal) を満足させる解の探索を始める．もし，すべてのグローバルゴールが満足されたならばネゴシエーションは終了する．

ここではエージェントはまず仮にサブゴールを選択し，そのサブゴールを満足するためにプランフラグメントを選び，それが必要としている資源を調べる．エージェント間制約が存在する資源を持つエージェントに対し，その資源使用の是非を確認するメッセージ (*Ok?* メッセージと呼ぶ) をそのサブゴールのグローバルプランの情報とともに送る．メッセージを受け取ったエージェントはエージェント間制約を満たすようにサブゴールを選択できるか調べ，可能ならば *Ok* メッセージを返し，だめならば *Conflict* メッセージとともにその競合情報，すなわち，*Ok?* メッセージを受け取った側のエージェントのサブゴールの排除集合を送る．このようにして，このフェーズでは探索と同時に排除集合の情報が次第に集められる．

エージェントが同じサブゴールを二度目に選択する時に前と同じ状況で試みても意味がないので他のエージェントの選択を強制的に変更させる必要がある．これがバックトラックに相当する．これは *Retry* メッセージにより実現され，一度選択されたサブゴールのプランフラグメントに対して割り当てられた資源を解放して，要求されたサブゴールを満足しようとする．

#### 4.2.2 協調探索フェーズ

協調探索フェーズ (Coordinated Search Phase) では起動エージェントは無効ゴール集合 (Nogood Goal Set) を計算する．そのために主要ゴール (p-goal) のゴール排除集合 (Goal Exclusion Set) を '完全' にする．ここで完全とはこれ以上値が変化しなくなるということで，この判断は他のエージェントにおける排除集合 (Exclusion Set) が完全になったという通知によって行なわれる．ゴール排除集合が起動エージェント間で交換されることによって，無効ゴール集合が求められ，同時に複数のグローバルゴールが満足されないことがわかった時は制約が強過ぎるということなので過制約解消フェーズへ移行する．



なお，もし，エージェントが完全なゴール排除集合を他のエージェントから受け取った時は非同期探索フェーズにいたとしても協調探索フェーズに移行する．ゴール排除集合は主要ゴールを持つすべての起動エージェントに送られるのではなく関連するエージェントにのみ送られるので協調探索フェーズに移行するエージェントを限定することが可能である．すなわち，ここでは相互関係のないグローバルゴールに関してはエージェントの間の協調はとらない．したがって，あるエージェントは非同期探索フェーズにあり，また他のエージェントは協調探索フェーズにあるということがありうる．

#### 4.2.3 過制約解消フェーズ

過制約解消フェーズ (Overconstrained Resolution Phase) では前のフェーズで求められた無効ゴール集合に基づいてあきらめるべきグローバルゴールを決定する．あきらめるゴールについて合意が得られた時は残りのグローバルゴールに関してどのグローバルプランを用いて解くかをゴール排除集合の情報に基づいて決定する．この段階でゴール排除集合，排除集合にはサブゴールに関して競合情報が集められているので，バックトラック無しに解を見つけることができる．

### 4.3 分散制約充足問題

#### 4.3.1 問題の表現

一般に制約充足問題は $m$ 個の変数 $x_1, x_2, \ldots, x_m$ とそれらの変数が取りうる値の領域 $D_1, D_2, \ldots, D_m$ および，その上で定義された制約 $P_k(x_{k_1}, x_{k_2}, \ldots, x_{k_j})$ が与えられた時にすべての制約を満たすように変数の値を決める問題と定義できる．分散制約充足問題は変数，制約が複数のエージェントに分散された問題として定式化できる．すなわち，変数があるエージェントに割り当てられ，エージェントは割り当てられた変数の値の決定を行なう．

### 4.3.2 非同期バックトラックアルゴリズム

分散制約充足問題のアルゴリズムはいくつか提案されているが，ここではその中でも基本的なものの一つである非同期バックトラックのアルゴリズム [64] を取り上げる．このアルゴリズムの基本的な考え方は次のとおりである (ここでは文献 [64] にならい一つのエージェントに対し一つの変数が割り当てられると仮定する)．まず，このアルゴリズムでは次のような前提条件を仮定している．

- エージェントに対してユニークな識別子 (id) が振られており，全順序がつけられている[1]．より大きい id を持つエージェントを高位のエージェントとする．
- 制約に対しそこに含まれる変数のエージェントのうち一番 id が大きいエージェントがその制約を評価する．制約に含まれる変数のエージェントから制約を評価するエージェントへ向かって論理的なリンクが張られている．

エージェントは自分に向かって張られたリンクを持つ他のエージェントの局所解の組合せ (Agent_View) と制約違反を起こす変数と値の組合せ (NoGood) を保持する．これらの値は最初は空である．

各エージェントは Agent_View 及び NoGood と矛盾しないように自分の変数の値を決めその値 (局所解) を外に向かって張られたリンク先のエージェントに対し送る (*Ok?* メッセージ)．このメッセージを受けとったエージェントは Agent_View を更新し，この新たな Agent_View と矛盾しないように自分の変数の値を決め，変更した時はその値 (局所解) を外に向かって張られたリンク先のエージェントに対し送る．もし，そのような変数の値が決められない時はその原因となる変数と値の組合せ (NoGood) を計算し，その NoGood 中の一番大きい id を持つエージェントに対し，その NoGood を送信し (*NoGood* メッセージ)，変数の値の変更を要求する．*NoGood* メッセージを受けたエージェントは自分の NoGood を更新し，新たに自分の変数の値を変更する．もし，それが不可能な場合はさらにその原因となる変数と値の組合せ (NoGood) を計算し，その NoGood 中の一番大きい id を持つエージェントに対し，その NoGood を送信する．NoGood は単調に増加するので無限ループを避けることができる．

[1]例えばエージェントの id を名前と見なして，その辞書式順序をとれば良い．



一つのエージェントに複数の変数が割り当てられる場合はエージェントに対して id を割り振るのではなく，変数に対して id を割り振り，変数単位に上で述べたアルゴリズムを動かせばよい．この時エージェント内でのメッセージのやりとりは手続きの呼び出しになる．また，NoGood に関しては単調に増加するので，エージェント内に一つだけ設け，エージェント内のすべての変数から共通にアクセスされるようにすることができる．

## 4.4 マルチステージネゴシエーションの分散制約充足問題へのマッピング

マルチステージネゴシエーションではグローバルゴール，グローバルプランといった概念を扱っている．そのため全体で複数のゴールがいくつかの方法で満足されるような問題のマッピングは自然な形で行なうことができる．さらに制約が強過ぎる時にどのグローバルゴールをあきらめればよいかなどアプリケーションから見て意味のある情報の抽出を解法の中に含めることが容易にできる．

しかし，それらのグローバルゴールなどの概念の意味を無視し，サブゴール間の関係を制約として表現することで，マルチステージネゴシエーションで扱っている問題を分散制約充足問題としてとらえることができる．この時，マルチステージネゴシエーションの問題を分散制約充足問題の一般的な形式にどのようにマッピングをとったらよいかを明らかにする必要がある．

分散制約充足問題は変数とその領域，制約，さらに変数がどのようにエージェントに分散されるかによって定義される．したがって，マルチステージネゴシエーションの問題が分散制約充足の変数，領域，制約，エージェントにどのようにマッピングしたらよいかを決める必要が出てくる．

### 4.4.1 変数とその領域

変数にはマルチステージネゴシエーションにおけるサブゴールを対応させる．この理由は次のとおりである．グローバルゴール，グローバルプランはエージェントにまたがって定義される (すなわち，グローバルに見て定義される) ものであるから，変数へのマッピングには馴染まない．一方サブゴール，プランフラグメントは一つのエー

ジェントに閉じているものである分散制約充足問題の枠組とも相性がよい．マルチステージネゴシエーションにおけるエージェント間の競合関係を表現する排除集合はサブゴールをベースとして定義され，基本的にはエージェントにおいてどのサブゴールを選択するかが他のエージェントとの関係で決定される．サブゴールが決定してしまえば，どのプランフラグメントを選択するかはエージェントの中に閉じることになる．もし，プランフラグメントを変数に割り当てると変数の数が必要以上に多くなり，適当ではない．

変数をサブゴールに対応させた場合，変数の領域はサブゴールを選択しない (これをNILで表すとする) か，選択する時はそのサブゴールに含まれるどのプランフラグメントを選択する (選択されるプランフラグメントをその値とする) かになる．すなわち，サブゴール$sg_{i,j,k}$に対して

$$sg_{i,j,k} \Leftrightarrow \bigvee_{l} pf_{i,j,k,l} \tag{4.1}$$

という関係がある時，これに対応した変数を$X_{sg_{i,j,k}}$とするとその領域は

$$\{\mathrm{NIL}, pf_{i,j,k,1}, \ldots, pf_{i,j,k,l_{i,j,k}}\} \tag{4.2}$$

となる．

これにより変数の値の選択はマルチステージネゴシエーションにおけるエージェント内で行なうことができるのでエージェントの対応は分散制約充足とマルチステージネゴシエーションとで同じとすることができる．

### 4.4.2 制約

マルチステージネゴシエーションでは制約としては資源の使用に関する制約として，エージェント内制約とエージェント間制約を考えた．これらの制約は分散制約充足へマッピングした場合，それぞれ次のように対応付けられる．

- エージェント内制約: 資源の容量の制約によりエージェント内で選択できるプランフラグメントに制限ができ，そのため，エージェント内で満足できるサブゴールに制約が生じることである．これは同時にはサブゴールに対応する変数の値がNILでない値がとれない組合せがあるということになる．ここで$V_{A,t}$をエージェント$A$において同時には満足できないサブゴールの集合を表すとする．



$t$はそのような集合が複数あることを示すための添字である．これを用いるとエージェント内制約は形式的には次のように書ける．

$$\neg \bigwedge_{sg_{i,j,k} \in V_{A,t}} (X_{sg_{i,j,k}} \neq \mathrm{NIL}) \tag{4.3}$$

- エージェント間制約: エージェント間において資源の使用に制約があるため，あるサブゴールを満足させた時に同時に満足させなければいけない (他のエージェントにおける，同じグローバルプランに属する) サブゴールがあることである．エージェント間制約があるサブゴール$sg_{i,j,k}$,$sg_{i,j,l}$間の制約は形式的には

$$\neg((X_{sg_{i,j,k}} \neq \mathrm{NIL}) \oplus (X_{sg_{i,j,l}} \neq \mathrm{NIL})) \tag{4.4}$$

と書ける．ここで$\oplus$は排他的論理和を表す．ここで排他的論理和を使ったのは両者のサブゴールが同時に満足されるかまたは満足されないかのどちらかであるからである．

さらに，マルチステージネゴシエーションでは暗黙のうちにすべてのグローバルゴールが満足されなければならないとしていたが，分散制約充足へマッピングするときにはこれを明示的な制約として表現する必要がある．ここでは，これをグローバル制約と呼ぶ．このグローバル制約はすべてのグローバルゴールに関してどれか一つのグローバルプランが満足されるということになる．あるグローバルプランのサブゴールのうち満足されるサブゴールが存在すれば上記のエージェント間制約により，そのサブゴールが属するグローバルプランは満足されることになる．したがって，グローバル制約は形式的には次のように書ける．

$$\exists f \exists k \; X_{sg_{i,j,k}} \neq \mathrm{NIL} \quad \text{for all } i \tag{4.5}$$

## 4.5 通信網のデータを用いた評価実験

ここでは，実際の応用に近い問題で評価するためNTTの通信網のデータより作成した通信網のパスの設定問題を例題としてとりあげた．評価実験の例題を述べる前にまず，簡単な通信網の例をとりあげ，それがどのようにマルチステージネゴシエーションと分散制約充足問題として考えることができるかを示す．

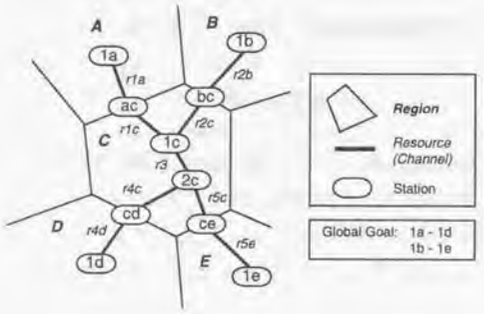



図 4.2: 簡単な通信ネットワークの例

### 4.5.1 簡単な通信網における例

図 4.2 に通信網の例を示す．ここでは$A$, $B$, $C$, $D$, $E$の五つの地域 (Region) に分かれており，中継点 (Station) として 1a, 1b, 1c, 2c, 1d, 1e がそれぞれ地域内にあり，また，ac, bc, cd, ce がそれぞれ地域間の境界点として存在する．図に示すように中継点間に通信路 (チャネル (Channel)) が存在し，これが資源 (Resource) に相当する．それぞれ容量は 1 であるとする．ここでのゴールは 1a と 1d 間，1b と 1e 間にそれぞれ容量 1 のパスを確保することである．この例では資源 (通信路) $r3$が両者のゴールを満足させるのに必要なため，両方のゴールを同時には満足できない[2]．

#### マルチステージネゴシエーションへのマッピング

上で述べた問題はマルチステージネゴシエーションでは次のように考えることができる．まず，それぞれの地域がエージェントに対応する．グローバルゴールはある地点からある地点へのパスを設定することであり，グローバルプランは表 4.1 のような中継点を通るものとなる．次に各エージェントにおけるサブゴールは表 4.2 に示すようになる．また，表 4.3 は各エージェントにおける資源とその総量，ならびにサブ

[2]実際の通信網では$r3$を二重化するなどの冗長構成をとることが考えられるが，ここでの例題はあくまでもマルチステージネゴシエーションの分散制約充足問題へのマッピングを説明するために非常に簡単化したものである．



表 4.1: 例におけるグローバルプラン

| Global Goal | Global Plan | Stations Used |
|---|---|---|
| $g1$ | $gp1$ | 1a - ac - 1c - 2c - cd - 1d |
| $g2$ | $gp2$ | 1b - bc - 1c - 2c - ce - 1e |

表 4.2: 例における各エージェントのサブゴール

| Agent | Global Goal | Global Plan | Subgoal | Stations to be Connected |
|---|---|---|---|---|
| $A$ | $g1$ | $gp1$ | $1a$ | 1a - ac |
| $B$ | $g2$ | $gp2$ | $1b$ | 1b - bc |
| $C$ | $g1$ | $gp1$ | $1c$ | ac - cd |
| | $g2$ | $gp2$ | $2c$ | bc - ce |
| $D$ | $g1$ | $gp1$ | $1d$ | cd - 1d |
| $E$ | $g2$ | $gp2$ | $1e$ | ce - 1e |

ゴールに対するプランフラグメントとプランフラグメントが使用する資源の量を示す．なお，この例では一つのサブゴールに対し一つのプランフラグメントが存在している．

ここでのエージェント内制約とは各通信路の容量の制約に基づくものである．エージェント$C$においてプランフラグメント$p1c$と$p2c$はともに資源$r3$を使用するため同時には選択できない．したがって，サブゴール$1c$と$2c$は同時には満足できないというエージェント内制約が存在することになる．

また，通信路という性格から地域間の境界点に達する通信路はその境界点の両側でそれぞれ通信路が割り当てられる必要がある．通信路が資源に相当することから，境界点をはさんで資源が同時に使用されるというエージェント間の制約があることになる．エージェント間の資源使用の制約からサブゴール間の制約が生じ，この例題ではエージェント間制約として，次のサブゴールの組はどちらか一方が満足されたら残り

表 4.3: 例における各エージェントのサブゴール，プランフラグメント，資源

| Agent $A$ | | Resource |
|---|---|---|
| Subgoal | Plan Fragment | $r1a$ |
| Resource Count | | 1 |
| $1a$ | $p1a$ | 1 |

| Agent $B$ | | Resource |
|---|---|---|
| Subgoal | Plan Fragment | $r2b$ |
| Resource Count | | 1 |
| $1b$ | $p1b$ | 1 |

| Agent $C$ | | Resource | | | | |
|---|---|---|---|---|---|---|
| Subgoal | Plan Fragment | $r1c$ | $r2c$ | $r3$ | $r4c$ | $r5c$ |
| Resource Count | | 1 | 1 | 1 | 1 | 1 |
| $1c$ | $p1c$ | 1 | | 1 | 1 | |
| $2c$ | $p2c$ | | 1 | 1 | | 1 |

| Agent $D$ | | Resource |
|---|---|---|
| Subgoal | Plan Fragment | $r4d$ |
| Resource Count | | 1 |
| $1d$ | $p1d$ | 1 |

| Agent $E$ | | Resource |
|---|---|---|
| Subgoal | Plan Fragment | $r5e$ |
| Resource Count | | 1 |
| $1e$ | $p1e$ | 1 |



のサブゴールも満足されなければならないことになる．

$$(1a, 1c), (1c, 1d), (1b, 2c), (2c, 1e) \tag{4.6}$$

また，ここでは$A$が$g1$の$B$が$g2$の起動エージェントとして考えることができる．

**分散制約充足問題へのマッピング**

先の例題の分散制約充足問題へのマッピングを行なう．

- 変数: マルチステージネゴシエーションにおけるサブゴールに対応して存在する．
- 変数の領域: 選択すべきプランフラグメント (この例では各サブゴールに対して一つ決まる) か，何も選択しない (NIL) かののどちらかになる．
- 制約: 次のような種類の制約になる．
  - エージェント内制約: プランフラグメント$p1c$と$p2c$は同時には選択できないため，エージェント$C$においてサブゴール$1c$ と$2c$は同時には満足できない．したがって，エージェント内制約は表 4.4 に示すようになる．
  - エージェント間制約: エージェント間で隣接する通信路が接続されなければならないという制約に対応し，具体的には表 4.5 に示すようになる．
  - グローバル制約: 各グローバルゴールに関してグローバルプランは一つずつ存在する．それらのグローバルプランから選んだサブゴールの一つに対応する変数がNIL以外の値をとらなければならない．この時，通信路の始点のサブゴールを選ぶことによってグローバル制約は始点のエージェントの中に閉じた制約とすることができる．例えば表 4.6 に示すようになる．

### 4.5.2 評価用例題の作成

ここでは NTT 伝送システム研究所で開発されたネットワーク構成管理データベース (CMDB: Configuration Management Database) [61, 62] のデータを利用して例題を作成した．このデータベースには NTT における通信網を反映した約 20 万本の通信路 (64K 回線相当で約 60 万回線分のデータ) が登録されている．

表 4.4: 例におけるエージェント内制約

| Agent | Constraint |
|---|---|
| $C$ | $\neg((1c \neq \mathrm{NIL}) \wedge (2c \neq \mathrm{NIL}))$ |

表 4.5: 例におけるエージェント間制約

| Global Plan | Constraint |
|---|---|
| $gp1$ | $\neg((1a \neq \mathrm{NIL}) \oplus (1c \neq \mathrm{NIL}))$<br>$\neg((1c \neq \mathrm{NIL}) \oplus (1d \neq \mathrm{NIL}))$ |
| $gp2$ | $\neg((1b \neq \mathrm{NIL}) \oplus (2c \neq \mathrm{NIL}))$<br>$\neg((2c \neq \mathrm{NIL}) \oplus (1e \neq \mathrm{NIL}))$ |

表 4.6: 例におけるグローバル制約

| Global Goal | Constraint |
|---|---|
| $g1$ | $1a \neq \mathrm{NIL}$ |
| $g2$ | $1b \neq \mathrm{NIL}$ |



このデータベースのデータより情報を抽出し，性能評価用の問題を作成した．ここではまずマルチステージネゴシエーションの枠組に基づいた問題を作成し，次に 4.4 節で述べた手法にしたがって分散制約充足問題への変換を行なった．使用したデータは基本的には交換局 (中継点に相当) とその局間を結ぶ通信路のデータである．ここで作成した問題は前章で述べた例題を複雑にしたものであり，例題の作成方針は以下の通りである．

- エージェント: 日本全国 10 の地域に分割し，各地域にエージェントを割り当てた．具体的には北海道，東北，関東，信越，北陸，東海，関西，中国，四国，九州の地域に分けた[3]．
- グローバルゴール: 通信路として 5760 回線の容量を持つ通信路を用いて各地域を代表する交換局を結ぶパスを求めることをグローバルゴールとする．各々の地域の中に代表となる交換局は単純にデータベース中の id の一番若いものを選んだ．ここで 5760 回線容量のものを選んだのはネットワークにおける主要なバックボーンのパスの設定を行なう問題とするためである． また，10 の地域に分かれることから ${}_{10}C_2 = 45$ 個のグローバルゴールが考えられる．この中からそれぞれの問題について異なる三つのグローバルゴールを設定した．
- グローバルプラン: パスの候補が (グローバルプランに相当) し，グローバルプランとしては代表点間を結ぶパスにおいて通過する交換局の数が最小となるものすべてを選んだ．
- サブゴール，プランフラグメント: グローバルプランからエージェントごとにサブゴールに分割した．また，ここではサブゴールに対してプランフラグメントを一つ対応させた．

この通信網の例題では制約が弱く，比較的容易に解が求まる場合と制約が強過ぎてすべてのグローバルゴールが満足できない場合の二通りが作成できる．ここでの実験に使用した問題の数は次の通りである．

- グローバルゴールがすべて満足できるもの: 131 個．
- すべてのグローバルゴールは満足できないもの: 24 個．

[3]この地域は東京支社を関東に含めた以外は NTT の各支社に対応している．

### 4.5.3 実験内容

上記方針に基づいてマルチステージネゴシエーションの枠組での問題を作成し，次に作成された問題を分散制約充足問題へと変換し，各々の問題について処理時間を測定した．処理時間はステージ数で測定した．ここでいうステージとは 1 ステージにおいて各エージェントがメッセージの種類によらず，一つのメッセージを処理することができる仮想的時間を表している[4]．したがって，エージェント内における処理に比べ通信に時間がかかるような場合，ステージ数を実際の処理時間の第一次近似としてとらえることができる．なお，エージェントは 1 ステージ内で複数のメッセージを他のエージェントに送ることは可能である．エージェントはそれぞれメッセージキューをもっており，受けとったメッセージは一旦キューにつながれ，各ステージにおいて一つずつメッセージがキューから取り出され，処理される．

非同期バックトラックのアルゴリズムは文献 [64] に基づいて実装した．そこでは一つのエージェントに対し，一つの変数が対応していることを前提としており，ここでの実験のように一つのエージェントに複数の変数が存在する場合に対処するために次のような実装を行なった．

- 一つのエージェント内に複数の変数が存在する場合，自エージェント内の変数に対するメッセージは前述したように仮想的に自分自身に対してメッセージを送るように実装したが，測定においてはステージ数においては仮想的なメッセージの処理に対しても単位ステージの処理がかかるとした．これはステージ数が実際の処理時間の近似としてとらえているからである．

- 非同期バックトラックにおいては変数の全順序をつける必要があるが，ここでは属するグローバルゴールでまず順序付けをし，その中では変数の対応するサブゴールのグローバルプラン (パスの候補) にしたがって順序付けをおこなった．これにより，同じグローバルゴールの変数は全順序をつけた中で固まって出現することになるので，あるグローバルゴールの中でのパスの探索は同じグローバルゴールの中で行なう可能性が高くなると考えられる．

[4]マルチステージネゴシエーションの “ステージ” とは異なる．



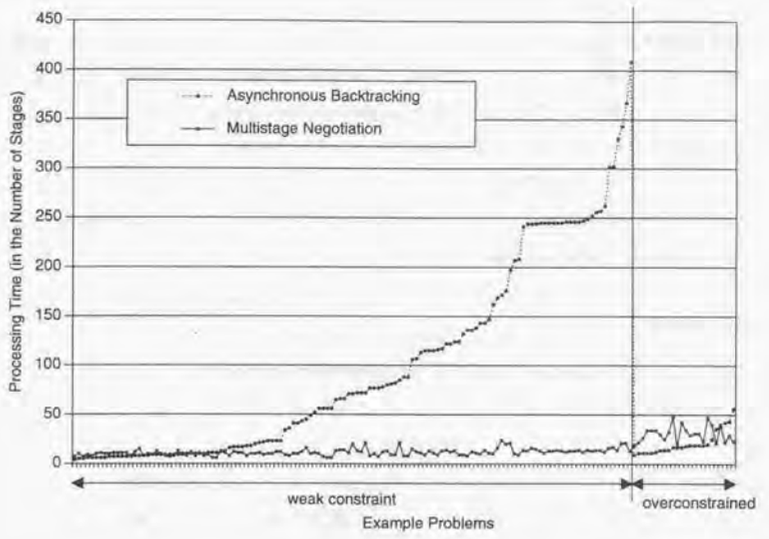



図 4.3: 処理時間 (ステージ数)

### 4.5.4 実験結果

図 4.3 にステージ数の測定結果を示す．横軸には例題をとり，縦軸には各々の問題に対するステージ数をとった．横軸は，まず制約が弱くグローバルゴールがすべて満足できるものについて非同期バックトラックにおけるステージ数の少ない順に問題を並べ，次に過制約の問題について非同期バックトラックにおけるステージ数の少ない順に問題を並べた．過制約の問題についてはマルチステージネゴシエーションでは過制約であることを認識する協調探索フェーズまでのステージ数をプロットしている．ただし，フェーズ自体はすべてのエージェントで一斉に切り替わるものではないので，ここではどれか一つのエージェントが次のフェーズに移行した時をもってフェーズが切り替わったとしている．

**すべての解が求まる問題**

最初の種類の問題に関してはマルチステージネゴシエーションではすべて非同期探索フェーズで解を見つけることが可能であった．非同期バックトラックでは一部はバックトラック無しに (すなわち，NoGood のメッセージが流されることなく) 解を見つけるものとバックトラックを含むものとが含まれている．

このグラフからわかるようにマルチステージネゴシエーションではステージ数にそれほど大きなばらつきがないものの，非同期バックトラックにおいては問題によってステージ数が大きく異なる結果が得られた．

**過制約の問題**

図 4.4 は図 4.3 の過制約の問題の部分を見やすくするために拡大したものである．ここでは図 4.3 と違ってマルチステージネゴシエーションに関してはフェーズごとのステージ数を示してある．また，過制約解消フェーズもあわせてプロットしている．過制約の問題においてはマルチステージネゴシエーションの方が非同期バックトラックに比較して多くの場合，過制約であることを認識するのにステージ数が多くなっていることがわかる．

### 4.5.5 考察

以上の実験結果より，3 フェーズプロトコルに関して次のようなことがいえる．

**性能上の比較 (非同期探索フェーズの効果):** 非同期分散バックトラックは基本的にはバックトラックはより高位の変数に対して行なわれる．バックトラックを行なう前にはそれより id の小さい変数の中ですべての組合せを尽くすことが必要となる．したがって，たまたま，高位の変数において後で制約違反を引き起こすような値の割当をした場合，バックトラックを起こすために大きな処理時間がかかってしまう．

一方，マルチステージネゴシエーションの非同期探索フェーズにおけるバックトラックに相当する処理では非同期バックトラックにおける id を意識しているわけでなく，競合が起こった時にローカルに競合を回避するように別のサブゴールが選択される．したがって非同期バックトラックのように高位の変数に対してバックトラックするために低位の変数であらゆる組合せを試すことはしない．非同期バックトラックに



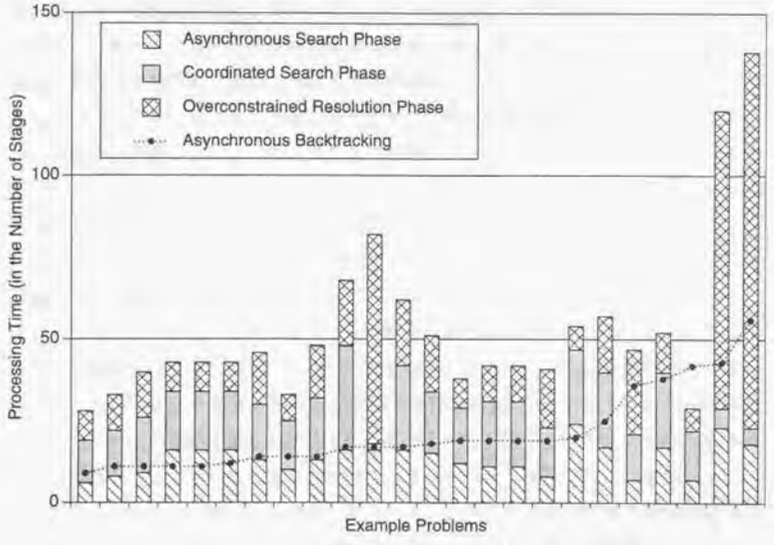



図 4.4: 処理時間 (ステージ数) – 過制約の場合

おいて高位の変数で制約違反を起こすような値に設定されてしまうとなかなか制約違反の原因となる変数に対しバックトラックがかからず，問題によって解をみつける時間に極めて大きなバラツキがでてしまったと解釈できる．したがって制約が比較的弱くすべての解が存在する時にはマルチステージネゴシエーションにおける非同期探索フェーズが効果を発揮することがわかる．

**過制約の問題における機能と性能のトレードオフ:** 非同期バックトラックにおいてはどこかのエージェントで空の NoGood が生成された段階で制約が強過ぎることが認識される．したがって，すべての制約違反の組を挙げるわけではない．一方，マルチステージネゴシエーションではすべての競合関係を調べた結果，過制約であることを認識している．すべての競合関係を調べたかどうかは無効ゴール集合が'完全' になったかで判断している．'完全' になったか判断するためにエージェント間で受取通知 (acknowledgement) のメッセージを交換している．すべての競合関係を調べることに加え，このメッセージ処理のオーバヘッドのため，過制約であることを認識するのに時間がかかっていると考えられる．

しかし，一方，マルチステージネゴシエーションにおいて得られる無効ゴール集合にはグローバルゴール間の競合関係の情報が入っているのでどのグローバルゴールをあきらめれば残りのグローバルゴールに関しては解くことができるかが無効ゴール集合を調べればすぐわかる．(分散制約充足問題においてはグローバルゴールという概念が無いので，どのグローバルゴールをあきらめたらよいかは自明ではない．) さらに，その時点において各グローバルプラン，サブゴールのレベルにおける競合に関しても情報が得られているのでどのグローバルゴールについて解くかを決定した段階でバックトラック無しに解を求めることが可能になっている．

したがって，マルチステージネゴシエーションにおいては過制約状態を認識するのに時間がかかるものの，一旦認識した後はどのグローバルゴールをあきらめたらよいかなどの情報が得られているという特徴がある．

## 4.6 まとめ

本章ではマルチステージネゴシエーションの枠組を分散制約充足問題として見た時，どのようにマッピングされるかを明らかにし，さらに，実際の通信網のデータを



もとにした例題を使って，マルチステージネゴシエーションと分散制約充足問題の基本的な解法である非同期バックトラックとの性能の比較を行なった．その結果，すべてのグローバルゴールの解が求まるような比較的容易な問題に対してはマルチステージネゴシエーションにおける非同期探索フェーズが効果を発揮し，非同期バックトラックに比べ，少ない処理時間 (ステージ数) で解を求めることができることがわかった．また，制約が強過ぎてすべてのグローバルゴールの解が求められないような問題に対しては，マルチステージネゴシエーションがすべての組合せの情報を集めていることになり，非同期バックトラックに比べ，多くの場合時間がかかる．しかし，非同期バックトラックで求められる NoGood の情報だけでは必ずしも，どのゴールをあきらめれば残りのゴールに関して解くことができるかはわからないというトレードオフがある．

# 第 5 章

# 市場モデルに基づく分散資源割当: 均衡的アプローチ

## 5.1 はじめに

本章ではマルチエージェントシステムにおいて市場モデルに基づいた資源割当について述べる．ここでは，他の市場モデルに基づいたアプローチと区別するために均衡的アプローチ (equilibratory approach) と呼ぶ．

第 3,4 章で述べたマルチステージネゴシエーションでは資源の割当を行なうか行なうかのいわばゼロ一の問題を扱っていた．したがって通信路の割当という例題の観点から考えると資源の量がうまく扱えない[1]．それに対して本章では，量を扱えるようなモデルの構築を目指す．

具体的には，マルチエージェントシステムにおける分散資源割当の問題を定式化し，そこに人間社会における (競争) 市場の考え方を導入することによって，資源の量を扱えるようにするようにする．

マルチステージネゴシエーションでは問題の構造に対応してエージェント間で交換すべき情報を規定した．これに対し，本章では市場の考え方を資源割当に導入することによって資源割当を資源の売買として実現する．エージェント間の通信は資源の売り手 (seller) と資源の買い手 (buyer) 間の通信に限定され，資源の売買のメッセージ

[1]ある単位量ごとに別の資源として扱うことでむりやり扱えるようにすることも考えられるが，ある量の割当を表現するのにいくつもの解の候補を考えることになり，余計な組合せを試みる必要が出てくるので効率的ではない．

の交換として規定されることになる．ここではエージェント間の情報の交換を価格情報 (売り手から買い手へ) と資源割当要求 (買い手から売り手へ) に限定し，エージェント間の明示的な協調のためのメッセージ交換をなくすことを目指す．

さらに，ここではエージェントは自分の効用関数 (utility function) を持ち，効用を上げるように行動を決定する．すなわち，エージェントが限られた局所的な情報をもとに自分の効用を上げるように振舞うことで，いかにシステム全体として適当な状態を達成できるかという問題になる．

## 5.2 マルチエージェントシステムにおける資源割当

ここで資源割当を資源 (resource) をアクティビティ (activity) に割り当てる問題と考える．マルチエージェントシステムにおける資源割当は資源とアクティビティをエージェントに対応づけることによって達成することができる．アクティビティに対応づけられたエージェント (アクティビティエージェント) は資源に対応づけられたエージェント (資源エージェント) に対して資源の割当要求を出す．それに対して資源エージェントは割当要求を認めるかどうか決定する．エージェントには効用関数が与えられ，各エージェントはそれぞれ効用を最大にするように振舞う．エージェントはシステムのグローバルな状態を必ずしも知らず，エージェントの限られた情報に基づいて行動を決定することになる．

マルチエージェントシステムにおける資源割当における問題は，“エージェントが限られたシステムに関する情報をもとに自分の効用を最大にするように行動を決めるときにシステム全体としてグローバルな目標を達成することができるか” ということである．本章で導入する市場モデルにおいては，エージェント間の資源の売買を通して資源はアクティビティに割り当てられる．アクティビティエージェントは資源の買い手となり，資源エージェントは資源の売り手となる．売り手のエージェントはその収入を最大にしようとし，買い手のエージェントはその支出を最小にしようとする[2]．本章のモデルにおいては買い手同士間は直接相互の通信は行なわない．したがって，買い手は他の買い手がどのような行動をとるかは直接には知らない．また，同様に売り手同士の相互の通信がなく，売り手は他の売り手がつけている資源の価格を知らないとする．エージェント間の通信は資源の割当要求 (買い手から売り手への通信)，実際に割り当てられる資源の量と価格の通知 (売り手から買い手への通信) に限られるとする．

[2]買い手の効用関数を例えば支出の逆数ないしは負となるように設定すると，支出を最小にすることは買い手の効用を最大にすることに相当する．



このようにエージェントはシステムに関する限られた情報に基づいて行動を決めるので，ヒューリスティクス (戦略) を導入することが必要になる．売り手の戦略はいかに対応づけられた資源の価格を決定するかに関わり，また，買い手の戦略はどの資源に対し，どれだけの量を要求するか決めることに関わることになる．

本章ではこのような考え方にもとづいた市場モデルに基づいた資源割当のアプローチについて述べ，その性質をシミュレーションを通して調べる [30]．具体的には通信ネットワークにおけるネットワーク制御に関する二つの例題を取り上げる．通信ネットワークは本質的に分散しており，多くの要素 (ネットワークノード，交換機，通信チャネルなど) が存在し，それらが独自の効用を持っていると考えられる．したがって，マルチエージェントシステムを考える上で有効なプラットフォームの一つを提供してくれると考えられる．

二つの例題のうち，例題 I では資源の価格の変化に対応してどれだけ資源の要求の変化させるかを決める感度系数について調べ，例題 II では売り手から買い手に提示される価格情報に関して通信遅延の影響について調べる．後で述べるように，通信遅延はシステムの振舞いに振動を引き起こすので，振動を押える手法についても言及する．

第2章で述べたように分散システムにおける資源割当に市場モデルの考え方を応用した例はいくつか報告されている [3]．そこでは CPU 時間や記憶領域が資源として扱われている [12, 44, 47, 59]．また，市場指向プログラミングの手法が提案されており輸送問題に応用されている [60]．これらの手法では資源の価格は競売機構を通して決められることが仮定されている．これに対し本章で提案する均衡的アプローチ (equilibratory approach) においては資源の価格はそれに対応する資源エージェントが資源の需要をもとに独自に決定する [29]．これにより入札機構に付随するオーバヘッドをなくすことを目標としている．

また，通信遅延が存在する場合のシステムの振舞いは計算生態学 (computational ecologies) [17] において調べられている．通信遅延が存在するとき，エージェントは古くなった情報をもとに行動を決定することになり，システムの振舞いは振動現象を示す [17]．このようなカオス的な振舞いを抑止するメカニズムも提案されている

[15]. 計算生態学では個々のエージェント戦略というよりむしろシステム全体としての振舞いに注目している. これに対し本章では具体的なエージェントの戦略がどうなるかについて考える.

## 5.3 市場モデルに基づいた分散資源割当

### 5.3.1 分散資源割当

ここでは $m$ 個の場所に存在する資源を $n$ 個のアクティビティに割り当てる資源割当問題を考える. ここで時刻 $t$ においてアクティビテイ $j$ が出す資源割当要求の総量を $R_j(t)$ で表す. ここでは $R_j(t)$ は時刻 $t$ によって変化するとする. $R_j(t)$ の中からアクテビティ $j$ は場所 $i$ の資源に対して $x_{i,j}(t)$ の資源割当要求を出すものとする. すなわち, $x_{i,j}(t)$ は時刻 $t$ におけるアクティビティ $j$ の場所 $i$ の資源に対する需要を表している. アクティビティ $j$ の資源割当要求の総和は $R_j(t)$ で表すことから, 次式が成立する.

$$\sum_{i=1}^{m} x_{i,j}(t) = R_j(t) \quad (1 \leq j \leq n) \tag{5.1}$$

ここで場所 $i$ の資源の総量を $N_i$ で表す. ここでは $N_i$ の値は一定であるとする. 場所 $i$ の資源に対する割当要求の総和は $N_i$ より小さくなるべきであるから次式が成立することが必要となる.

$$\sum_{j=1}^{n} x_{i,j}(t) \leq N_i \quad (1 \leq i \leq m) \tag{5.2}$$

ここでは $N_i$ の値は十分大きく, 上の制約はいつも満足されると仮定する. 言い換えれば資源の割当要求は常に満たされるとする. したがって場所 $i$ の資源に対する割当要求 $x_{i,j}(t)$ は実際に割り当てられる資源の量をも表すことになる.

ここで, 場所 $i$ の資源の時刻 $t$ における使用率を $u_i(t)$ で表すとする. $u_i(t)$ は次式で定義される.

$$u_i(t) = \frac{\sum_{j=1}^{n} x_{i,j}(t)}{N_i} \tag{5.3}$$

ここではグローバルな目標として異なる場所における資源の使用率を均等化することを考える. 本章の後半で述べるシミュレーションにおいては資源の使用率の分散 $V_u(t)$ を用いて異なる場所の資源の使用率の均等化が実現できるかを表現することにする. $V_u(t)$ は次のように与えられる.



$$V_u(t) = \frac{\sum_{i=1}^{m}(u_i(t) - (\sum_{i=1}^{m} u_i(t))/m)^2}{m} \tag{5.4}$$

すなわち, グローバルな目標は $V_u(t)$ の値を最小化することになる.

### 5.3.2 市場モデル

次に市場モデルを導入する. それぞれの場所ごとに売り手が存在し, 売り手が自分の資源に対して価格をつける. 場所 $i$ の資源の時刻 $t$ における単位量当たりの価格を $p_i(t)$ で表すことにする. 場所 $i$ における売り手は過去の需要 $(\sum_{j=1}^{n} x_{i,j}(t')\ (t' < t))$ に基づいて資源の価格 $p_i(t)$ を決定する. 買い手 $j$ は場所 $i$ の資源に対する時刻 $t$ における割当要求を過去の価格 $(p_i(t')\ (t' < t))$ に基づいて決める. ここでのモデルでは買い手同士, 売り手同士の直接の通信は無いものとする. したがって, 売り手は価格を決定する際に自分の場所以外の資源の価格は知らない. また, 買い手は資源の割当要求を出す際に他の買い手がどのような資源の割当要求を出すかを知らないとする.

#### 売り手の効用

売り手の目標はその収入を最大にすることである. ここで場所 $i$ の資源に対応づけられた売り手 $i$ の時刻 $t$ における効用 $U_i^s(t)$ を次のように定義する.

$$U_i^s(t) = p_i(t) \sum_{j=1}^{n} x_{i,j}(t) \tag{5.5}$$

#### 買い手の効用

買い手の目標はその支出を最小にすることである. アクティビティ $j$ に対応づけられた買い手 $j$ の時刻 $t$ における支出は $\sum_{i=1}^{m} p_i(t)\ x_{i,j}(t)$ で与えられる. ここで買い手の効用を支出の負の値とする. すなわち, 買い手 $j$ の効用 $U_j^b(t)$ は次のようになる.

$$U_j^b(t) = -\sum_{i=1}^{m} p_i(t)\ x_{i,j}(t) \tag{5.6}$$

したがって買い手の効用を最大にすることは支出を最小にすることになる.

売り手の効用は買い手の資源割当要求 $(x_{i,j}(t))$ によって変わってこよう. 同様に買い手の効用は売り手が設定する価格 $p_i(t)$ に影響される. しかし, 買い手と売り手の

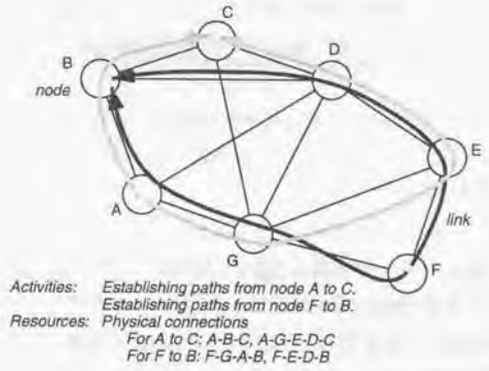



図 5.1: 通信ネットワークにおけるパス割当の例

間の通信は価格の通知 (売り手から買い手), 資源の割当要求 (買い手から売り手) に限られており, 買い手 / 売り手がその効用を最大化するにはヒューリスティクス (戦略) に頼ることになる.

## 5.4 例題 I – 感度系数の影響

本節では通信ネットワークにおいてパスを設定する例を考える. ここでは $n$ 個の出発地点と目的地点の組を考える. $j$ 番目の組 ($j = 1, 2, \ldots, n$) について $l_j$ 個の物理的な経路が存在し, この経路を使ってパスを設定する. 各経路は複数のリンクから構成されることもあり, また, 同一のリンクが複数の経路で共有されることもありうる. 例を図 5.1 に示す.

本節ではアクティビティ $j$ は $j$ 番目の出発 / 目的地点の組に関するパスを設定することに対応する. 場所 $i$ における資源は物理的なコネクション $i$ ($i = 1, 2, \ldots, m$) の帯域幅に対応する. 帯域幅が $N_i$ で表されることになる.

$j$ 番目の出発 / 目的地点の組に対するパスを設定するとき, パスの必要な帯域の和は時刻によらず一定で, $R_j$ ($j = 1, 2, \ldots, n$) で表されるとする. また, $j$ 番目の出発 / 目的地点の組に対する物理的なコネクションの集合を $\mathcal{S}_j$ で表すとする ($|\mathcal{S}_j| = l_j$).



アクティビティ $j$ はこれらの物理的なコネクションに関して資源の割当要求を出すものとする. すなわち,

$$\begin{cases} x_{i,j}(t) \geq 0 & i \in \mathcal{S}_j \\ x_{i,j}(t) = 0 & i \notin \mathcal{S}_j \end{cases} \tag{5.7}$$

グローバルな目標は異なる物理的なコネクションの使用率を均等化することになる.

以下のシミュレーションでは時間は時間スロットに区切られたものとして扱う. 買い手と売り手間の通信は各時間スロットにおいて資源割当の対象となるコネクションとは別の (信号線専用の) 通信チャネルを用いて行われることを仮定する. さらにここでは通信において遅延はないものとする. 買い手は現在の時間スロットにおける資源の価格をもとに資源割当要求を出し, 売り手は各時間スロットにおける資源の価格をそれ以前の時間スロットにおける資源の需要をもとに決定する.

### 5.4.1 エージェントの戦略

#### 売り手の戦略

売り手の目標はその売上を最大にすることである. その戦略として, 対応する資源 (物理的なコネクション) の需要が大きくなれば価格を上げ, 需要が小さくなれば価格を下げることが考えられる. コネクション $i$ の時間スロット $t$ における需要は (1) コネクション $i$ の (全部ないし一部) をすでに使用している帯域幅と (2) 新たに割当要求のあった帯域幅の和となる. ここでは割当要求はすべて満たされるという仮定 (5.2式) をおいているので, 新たな割当要求はすべて満たされ, 実際にアクティビティに割り当てられる資源の量は割当要求の総和と同じとなる. したがって場所 $i$ における資源の使用率 $u_i(t)$ は対応する資源に対する資源割当要求を用いて次のように書ける.

$$u_i(t) = \frac{\sum_{j=1}^{n} x_{i,j}(t)}{N_i} \tag{5.8}$$

以下のシミュレーションでは簡単化のために資源の使用率 $u_i(t)$ を場所 $i$ の資源の価格として表す. すなわち,

$$p_i(t) = u_i(t) = \frac{\sum_{j=1}^{n} x_{i,j}(t)}{N_i} \tag{5.9}$$

とする.

**買い手の戦略**

時間スロット $t$ において買い手 $j$ は資源割当要求 $x_{i,j}(t)$(すなわち，コネクション $i$ に対する帯域幅の要求) を価格 $p_i(t)$ に基づいて決めることになる．ここで $S_j$ の中で一番安いコネクションを $k$ で表す．時刻 $t$ におけるコネクション $k$ に対する資源割当要求 $x_{kj}(t)$ を次のように決める．

$$x_{kj}(t) = x_{kj}(t-1) + \alpha(R_j - x_{kj}(t-1)) \tag{5.10}$$

ここで，$\alpha$ $(0 \leq \alpha \leq 1)$ は感度係数を表す．$\alpha = 0$ のときは 5.10 式は $x_{kj}(t) = x_{kj}(t-1)$ となり，時間スロット $t$ における値はその直前の時間スロットにおける値と同じになる．また，$\alpha = 1$ のときは 5.10 式は $x_{kj}(t) = R_j$ となり，すべての資源の割当要求はコネクション $k$ に対して行なわれることになる．他のコネクション $i \in S_j$ $(i \neq k)$ については次式にしたがって割当要求が決定される．

$$x_{ij}(t) = (1-\alpha) \times x_{ij}(t-1) \quad (i \neq k) \tag{5.11}$$

ここで割当要求の総和 $(\sum_{i=1}^{m} x_{ij}(t))$ はアクティビティが必要とする資源の量 $(R_j)$ と等しくなる．

### 5.4.2 シミュレーションのパラメータ

以下のシミュレーションにおいてはパラメータの値を次のように決めた．資源 (コネクション) の種類の数は 100 $(m = 100)$，また，アクティビティ (出発 / 目的地点の組) の数も同様に 100 $(n = 100)$．各アクティビティが必要とする資源の量は一定で 50,000 $(R_j = 50,000)$．場所 $i$ における資源の量 (コネクション $i$ の帯域幅に相当) $N_i$ は 1 と 1,000,000 の間の一様分布とする．したがって $N_i$ の平均値 $(\overline{N_i})$ は約 500,000 となる．システムに存在する資源の総量は $\overline{N_i} \times m = 500,000 \times 100$ となり，また，資源の割当要求の総和は $R_j \times n = 50,000 \times 100$ となる．したがって資源の使用率の平均はおよそ $0.1(= (R_j \times n)/(\overline{N_i} \times m))$ となる．ここでは簡単のためそれぞれの出発 / 目的地点の組に関して 5 個のコネクションが利用可能とする (すべての $j$ に対して $l_j = 5$)．同一のコネクションが異なる出発 / 目的地点間の組で共有されていることもある．



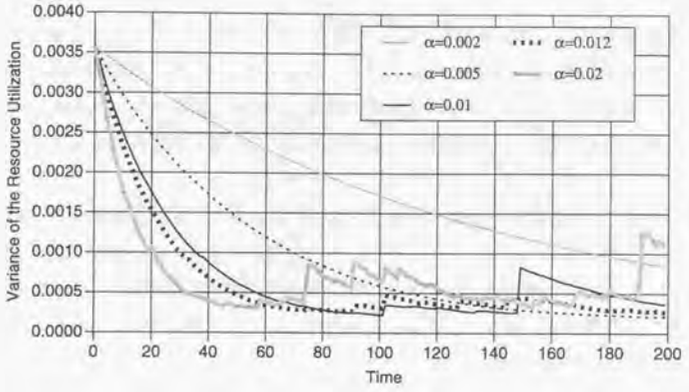



図 5.2: 資源の使用率の分散 $(V_u)$ (例題 I)

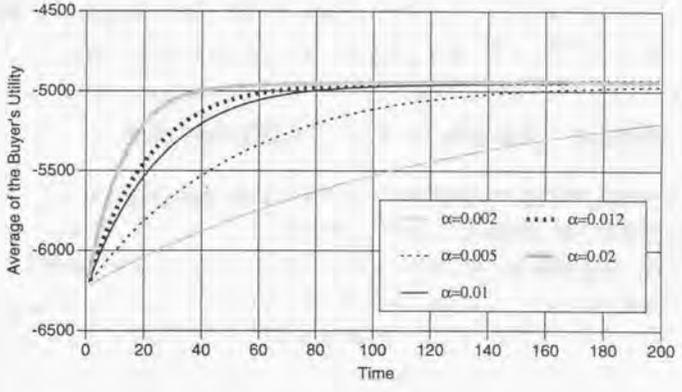



図 5.3: 買い手の効用の平均 (例題 I)

### 5.4.3 シミュレーション結果

資源の使用率の分散を時間スロットの関数としてプロットしたものを図 5.2 に示す．ここでは感度定数 $\alpha$ の値を 0.002 から 0.02 まで変えている．この図からわかるように $\alpha$ の値が大きくなると資源の使用率の分散はより早く減少する．これは $\alpha$ の値が大きい場合は資源割当における変化も大きくなり，より早く収束するからと考えられる．

また，図 5.2 は資源の使用率の分散があるレベルに達すると，使用率の分散が振動を始めることを示している．これは次のように説明できよう．安い資源に対する需要が大きくなると，その価格が上昇する．価格が上昇すると需要が減少し，価格が下がる．また，需要が元に戻る．このような繰り返しによって図 5.2 に見られるような振動現象が起こると考えられる．

ここで $\alpha$ の値が小さいとき (図 5.2 で 0.005 以下のとき)，振動現象は無視できるくらい小さく，資源の使用率の分散は収束する．これがここでいう資源の市場の均衡点になっているといえる．そこでは資源はほぼ均等に使用されていることになる．

また，買い手の効用の平均 $U^b$ の時間的な変化を図 5.3 に示す．この図からわかるように $\alpha$ が大きくなると $U^b$ も同様に大きくなる．これは $\alpha$ の値が大きいときは買い手にとって有利な結果になることを示している．これは買い手はできるだけ安い資源を使うべきという直観と合致する．

## 5.5 例題 II – 価格情報の伝達における通信遅延の影響

第 2 の例題 (例題 II) では価格情報の伝達における通信遅延の影響を調べる．ここでは売り手が買い手に価格情報を定期的に時間間隔 $T$ でブロードキャストするものとする．また，価格情報が買い手に到達するまでに遅延 $D$ が存在するものとする．

この例題ではチャネルには発生した呼びが割り当てられる [49]．ここでは $n$ 個の呼源 ($n$ アクティビティ) が存在し，そこから発生する呼びを $m$ 個のチャネル ($m$ 個の資源) のどれかに割り当てる．呼びにはチャネルの帯域の一部が割り当てられる (図 5.4)．

チャネル $i$ の帯域幅が場所 $i$ の資源に対応し，チャネル $i$ の帯域幅の大きさが場所 $i$ の資源の量 $N_i$ に対応する．チャネル $i(i = 1, 2, \ldots, m)$ に対して，売り手 $i$ が対応づけられ，また，呼源 $j(j = 1, 2, \ldots, n)$ に買い手 $j$ が対応づけられる．買い手は新たに発生した呼びのためにチャネルの帯域幅を購入する．ここで買い手 $j$ が時刻 $t$ において必要とする資源の総量を $R_j(t)$ で表す．$R_j(t)$ は呼源 $j$ から発生している呼びと時刻 $t$ において呼源 $j$ から新たに発生した呼びの総和となる．



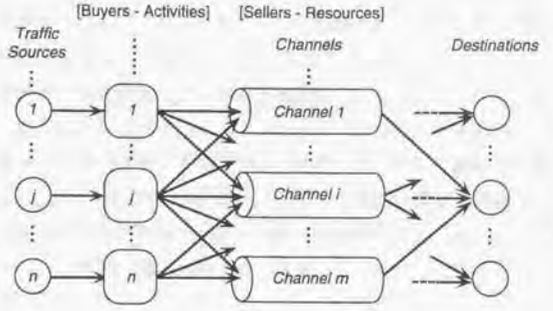



図 5.4: チャネル割当のモデル

ここでは各呼びを収容するために必要な帯域幅は固定とする．ただし，呼びの発生する個数が時間とともに変動するため $R_j$ の値は時間とともに変動することになる．

### 5.5.1 エージェントの戦略

#### 売り手の戦略

例題 I と同様にチャネル $i$ の帯域幅の価格をチャネル $i$ の使用率と設定した．すなわち，

$$p_i(t) = u_i(t) = \frac{\sum_{j=1}^{n} x_{i,j}(t)}{N_i} \tag{5.12}$$

となる．

**買い手の戦略**

ここでは価格情報が $T$ の一定間隔でブロードキャストし，また，価格情報の伝達に $D$ の遅延が存在することを仮定している．そのため買い手は現時点の正確な価格を必ずしも知ることはできない．ここで時刻 $t$ について一番最後に価格情報がブロードキャストされた時刻を $\langle t \rangle$ で表す．すなわち，$\langle t \rangle = \lfloor t/T \rfloor T$ となる．ここで $\lfloor x \rfloor$ は $x$ を越えない最大の整数を表す．買い手にとって時刻 $t$ において知りうる最新の価格は時刻 $\langle t - D \rangle$ における資源の価格，すなわち，$p_i(\langle t - D \rangle)$ $(i = 1, 2, \ldots, m)$ となる．そこで，買い手は $p_i(\langle t - D \rangle)$ の情報から現在の価格を推定することを考える必要がある．ここでは，その推定の手法として次の二つの手法を考える．

**0 次推定** 時刻 $t$ における場所 $i$ の資源の推定価格を $\tilde{p}_i(t)$ で表す．0 次推定では時刻 $t$ における場所 $i$ の資源の価格は時刻 $\langle t - D \rangle$ における価格から変わらないものとする．すなわち，

$$\tilde{p}_i(t) = p_i(\langle t - D \rangle) \tag{5.13}$$

となる．

**1 次推定** 1 次推定では $(\langle t - D \rangle, t)$ の間の価格の変化率が $(\langle t - 2D \rangle, \langle t - D \rangle)$ の間の価格の変化率から変化しないものとする．すなわち，

$$\tilde{p}_i(t) = p_i(\langle t - D \rangle) + \{p_i(\langle t - D \rangle) - p_i(\langle t - 2D \rangle)\} \frac{t - \langle t - D \rangle}{\langle t - D \rangle - \langle t - 2D \rangle} \tag{5.14}$$

となる．

新たな呼びが発生したとき，チャネルの帯域幅がこの呼びに対して割り当てられることになる．ここで時刻 $t$ において呼源 $j$ から新たに到着する呼びの量を $\Delta R_j^+(t)$ で表すものとする．同様に呼びが時刻 $t$ で終了したときには割り当てられていた帯域幅を解放することになる．ここで時刻 $t$ において解放される帯域幅の量を $\Delta x_{i,j}^-(t)$ で表す．添字の $i$ はこの帯域幅がどの呼源から発生した呼びに対して割り当てられたかを表している．

ここで呼源 $j$ からの呼びが時刻 $t$ で終了したとする．さらにこの終了した呼びがチャネル $i$ に割り当てられていたとする．このとき時刻 $t$ においてこの呼びが解放する資源の量を $\Delta x_{i,j}^-(t)$ で表す．さらに時刻 $t$ において呼源 $j$ で新たな呼びが発生し，それがチャネル $i$ に割り当てられたとする．このとき場所 $i$ の資源 (チャネル $i$) に対する割当要求 $x_{i,j}(t)$ は

$$x_{i,j}(t) = x_{i,j}(t^-) + \Delta R_j^+(t) - \Delta x_{i,j}^-(t) \tag{5.15}$$

と書ける．ここで $t^-$ は時刻 $t$ の直前の時刻を表す．買い手の戦略は支出を最小にすることであるから，買い手 $j$ は現在の資源の価格を推定し，新たな呼び $(\Delta R_j^+(t))$ に対しては一番安いと判断する資源に対して資源の割当要求をだす．ここで場所 $k$ の資源が時刻 $t$ において一番安い資源と買い手が推定したとしよう．買い手 $j$ の場所 $k$ の資源に対する割当要求は

$$x_{k,j}(t) = x_{k,j}(t^-) + \Delta R_j^+(t) - \Delta x_{k,j}^-(t) \tag{5.16}$$

となる．また，これ以外の資源 $i(i \neq k)$ に関しては時刻 $t$ における割当要求は

$$x_{i,j}(t) = x_{i,j}(t^-) - \Delta x_{i,j}^-(t) \quad (i \neq k) \tag{5.17}$$

となる．ここで買い手は一番安いと推定する資源 $(k)$ からのみ新たな資源の割当要求を出すので，$\Delta R_j^+(t)$ の値は $k$ 以外の $i$ についてはゼロになる．

### 5.5.2 シミュレーションのパラメータ

以下のシミュレーションでは，二つの呼源 $(n = 2)$ と二つのチャネル $(m = 2)$ が存在する場合を考える．それぞれのチャネルの帯域幅は 156 M bits/sec (ATM (Asynchronous Transfer Mode) ネットワークにおける典型的な帯域幅に相当) とする．資源の単位を 1 kbits/sec とすると 156 Mbits/sec の帯域幅は資源の総量が 156,000 であることに対応する $(N_i = 156,000)$．呼びはそれぞれの呼源から 0.14 秒に 1 個の割合でポワソン過程にしたがって発生するものとする．また，呼びの保留時間は平均 60 秒の指数分布にしたがうものとする．これらの値は典型的な電話の呼びに対応する．それぞれの呼びは 64 kbits/sec の帯域を使用するものとする (資源の使用量は 64 となる)．これは PCM (Pulse Code Modulation) によって符合化された音声に相当する．

### 5.5.3 シミュレーション結果

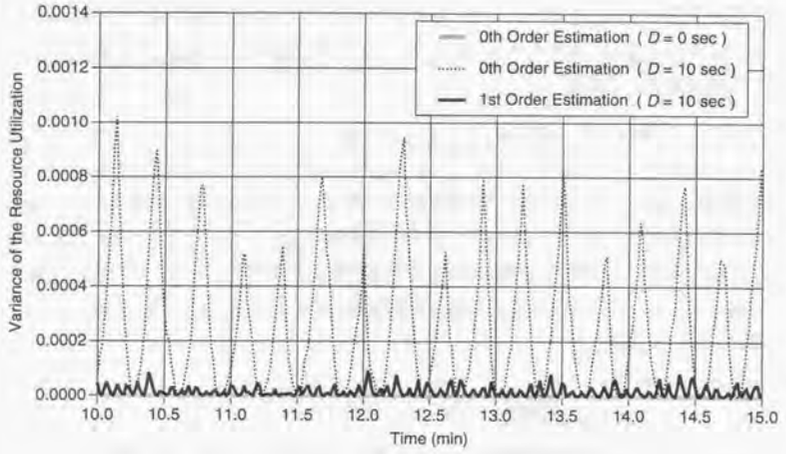



図 5.5: 資源の使用率の分散 ($V_u$) (例題 II)

図 5.5 と図 5.6 においては価格情報がブロードキャストされる間隔は $T$ は十分小さいものとする．図 5.5 は資源の使用率の分散 $V_u$ の時間的な変化を示している．この図には 0 次推定の場合と 1 次推定の場合が含まれている．シミュレーションの結果においては最初の 10 分間はシステムの過渡的な状態を除くため省いてある．この図に示すとおり通信遅延が 0 sec の場合は資源の使用率の分散 $V_u$ はほとんどゼロである．これは買い手が一番安い (すなわち一番使われていない) チャネルを正しく選択することができ，チャネルの使用率のバランスが崩れたとしても直ちに修正することができるからである．しかし，通信遅延が存在するとき ($D = 10$ sec) は，0 次推定を用いた場合，分散 $V_u$ の値が振動する．これは買い手が古くなった価格情報をもとに割り当てるべきチャネルを選択するからである．これに対し，1 次推定が使われた場合はシミュレーションの結果によると通信遅延がある場合でも振動の度合いは少なくなっている．これは遅延時間 $D$ の間の $R_j(t)$ の変化が小さいときは 1 次推定の効果が発揮されることを示している．

図 5.6 は買い手の効用 $U^b$ ($= \frac{1}{2}\sum_{j=1}^{2} U_j^b$) の平均を示している．二つの推定手法における効用 $U^b$ には大きな違いはなかった．このことは推定手法の違いは資源の使用率



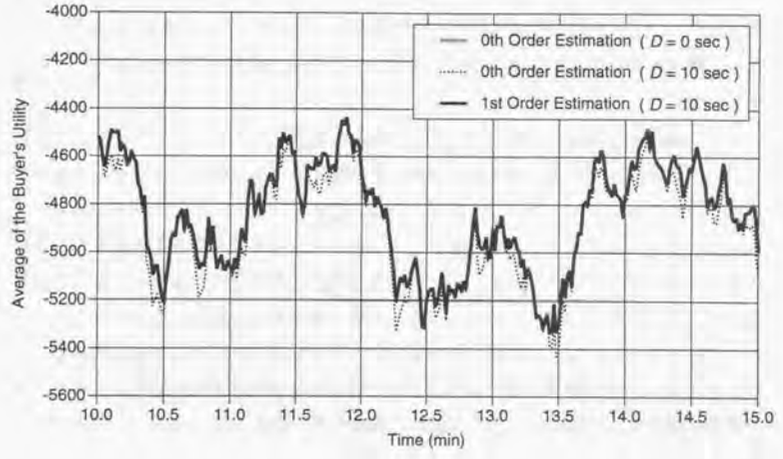



図 5.6: 買い手の効用の平均 (例題 II)

のバランスに効果を発揮することを示している.

## 5.6 まとめ

本章でマルチエージェントシステムにおける分散資源割当について市場モデルに基づいた試みを述べた. 通信ネットワークを題材とした二つの例題を取り上げ, シミュレーションを用いた評価を行なった. シミュレーション結果はここで提案した手法が資源の使用率の分散に振動を起こすことを示している.

この振動現象は大きな感度係数 (例題 I), または価格情報の伝達における遅延 (例題 II) によって引き起こされている. 感度係数を小さくすること (例題 I) によって, または推定手法を導入すること (例題 II) によって振動の度合いを小さくすることができる.

ここで提案した手法は, まだあくまでも試みの段階にあり, より現実に即した状況においても実験を進めるとともに, 詳細な解析的なモデルを構築をすることは今後の課題の一つである.

また, 価格の決め方によりバラエティを持たせることも考えられる. 本章では単純に資源の使用率を価格としていたが, 売り手が独自に価格を設定することも考えられる [29]. 価格の設定の仕方によりシステム全体の振る舞いも変わってこよう. また, ここでのシミュレーション実験では資源の量を中心に考え, 質まで考慮に入れなかった. エージェントの効用に質を反映させる (例えばある資源の割当に対してはより高い効用を与える) ことにより, 同じ枠組で資源の質も扱えるようにすることができよう.

# 第 6 章

# 協調プロトコル記述言語 AgenTalk

## 6.1 はじめに

マルチステージネゴシエーションでは問題の構造に対応した情報の交換を規定した. また, 市場モデルでは売り手, 買い手という構造を仮定し, その構造に基づく情報の交換を規定した. このような協調メカニズムを用いて実際のマルチエージェントシステムを構築していくためには, エージェント間のインタラクションにおけるアプリケーションレベルのプロトコル (協調プロトコル) を記述できるような枠組が必要となる. このとき種々多様なアプリケーションを実現しようとするとそれぞれの応用領域に適した協調プロトコルが必要になろう. さらにユーザの代わりとなって働くエージェントの実現では, ユーザ自らプロトコルを記述することも考えられる. このためには, プロトコルを一から記述していくのではなく, 既存のプロトコルをベースにして段階的にプロトコルを定義することにより, 応用に適したプロトコルを容易に実現できることが望まれる.

第 2 章で述べたようにいくつかの協調プロトコル記述言語が提案されているが, これらの言語では協調プロトコルを段階的に定義することを言語機能としてサポートしていない. そこで, 本章ではプロトコル記述にオブジェクト指向言語に見られるような継承機能を導入し, 段階的なプロトコル定義を可能にした協調プロトコル記述言語 AgenTalk を提案する [31, 32, 33, 34, 37]. AgenTalk の段階的なプロトコル定義機能により, あらかじめ設計したプロトコル群をライブラリーとして保存しておき, 新たなプロトコルを記述する時は既存のプロトコルとの差分だけを定義すればよくなる. 特にマルチエージェントにおける種々の協調プロトコルを実験, 実装するにあたって

は継承機能を用いて定義することにより，プロトコルの体系化を行なうこともできると考えられる．

本章では，まず AgenTalk の設計方針を述べ，次にその機能を説明する．さらに記述能力を検証するために契約ネットプロトコル [55] を記述し，さらにそれをベースに第 3,4 章で述べたマルチステージネゴシエーション [6] におけるような多段階の交渉を可能とするように拡張する．さらに，拡張機能として AgenTalk の言語機能を用いたメタレベル制御の考え方 [28] についても述べる．

## 6.2 設計方針

AgenTalk は協調プロトコルの形式的仕様を定義するというよりも，むしろ実際に動くプログラムの記述を目標にしている．さらに，エージェント自体がさまざまな枠組 (例えば黒板モデル，プロダクションルール) で記述できるように，エージェントの実現方法までは深く関わらず，プロトコル記述に着目する立場をとっている．

以下，協調プロトコルを記述する上で必要な機能とそれに対する AgenTalk の設計方針を述べる．

1. プロトコルにおける状態の明示的な記述

   第 3,4 章で述べたマルチステージネゴシエーションの研究において，エージェント間のプロトコルをエージェント間で交換するメッセージのクラス (契約ネットプロトコルにおけるタスク公告など) とそのメッセージを受け取った時にエージェントがとるべき行動 (メッセージハンドラ) により定義した．これを一般のオブジェクト指向言語による記述と対応させると，メッセージの種類を決め，それに対するメソッドを定義することに相当する．しかし，これだけでは，協調プロセスにおけるエージェントの状態を明示的には記述できないため，メッセージハンドラを定義・拡張することは容易ではなかった．協調プロトコルの記述としては協調プロセスにおけるエージェントの状態を明示的に表現できることが必要だと考えられる．

   そこで，AgenTalk では通信プロトコル記述でよく用いられている拡張有限状態機械 (Extended Finite State Machine)[16] をベースに協調プロトコルを記述することにした．拡張有限状態機械は有限状態機械に変数を許すように拡張したものである．ここではこの有限状態機械の表現をスクリプト (script) と呼んでいる[1]．

2. エージェントごとのプロトコルの容易な特化

   エージェントは，基本的には共通のプロトコルにしたがうものの，部分的にエージェント独自の振舞いをすることが必要になる．例えば契約ネットプロトコルでは，具体的なタスク公告の内容はエージェントごとに異なってこよう．

   そこで，AgenTalk ではエージェント関数というスクリプトを特化するためのインタフェースを設けることにした．エージェント関数とは後述するようにスクリプト内の状態遷移規則から呼び出される一種のコールバック関数で，エージェントが独自に定義できるものである．エージェント関数を用いてエージェント独自の振舞いを記述することが可能になる．

3. エージェント内の複数協調プロセスの実現

   例えば，契約ネットプロトコルにおいてエージェントが複数のタスク公告に対し同時に入札する場合を考えよう．この時，エージェントは複数の協調プロセスに係わっていることになる．すなわち，エージェント内に複数の協調プロセスを表現できることが必要になる．

   AgenTalk では，複数のスクリプトを一つのエージェント中に動作させることで，複数の協調プロセスを一つのエージェント内に実現することにした．

以上に加えて，AgenTalk では段階的なプロトコル定義を実現するためにオブジェクト指向言語にみられる継承の考え方を導入している．具体的にはスクリプト間で継承を実現し，既存のスクリプトを拡張する形で新たなスクリプトを定義できるようにしている．AgenTalk では状態を明示的に定義できるため，スクリプトの継承においては，状態を単位とした継承が容易に実現できる．この点が一般のオブジェクト指向言語におけるメソッドなどの継承とは異なる点である．

[1] エージェントがプロトコルにしたがって行動するのを演劇において俳優が台本 (スクリプト) にしたがって演じるということになぞらえている．

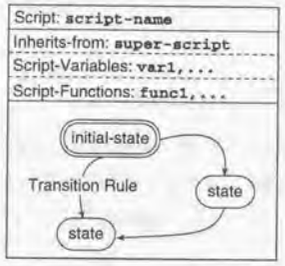



図 6.1: スクリプトの構成要素

## 6.3 基本機能

### 6.3.1 スクリプトの構成

前述のように AgenTalk では一つの拡張有限状態機械の表現をスクリプト (script) と呼んでいる．スクリプト $SC_i$ は，状態の集合 $S_i$，初期状態 $so_i \in S_i$，状態遷移規則の集合 $R_i$，変数 (スクリプト変数と呼んでいる) の集合 $V_i$ に加えて，スクリプトにローカルな関数 (スクリプト関数と呼んでいる) の集合 $F_i$，継承先のスクリプト $I_i$ の組で表現される (図 6.1)．すなわち，$SC_i =< S_i, so_i, R_i, V_i, F_i, I_i >$ となる．

ここで継承の対象となるのは，スクリプトの構成要素である状態，初期状態，状態遷移規則，スクリプト変数，スクリプト関数である．継承された定義はあたかも自分のところで定義されたかのように扱われる．継承の具体例は後の例題で説明する．

### 6.3.2 エージェントとスクリプト

エージェントのプログラムから見ると，スクリプトは一種の手続き群としてみることができる．エージェントがある協調プロトコルにしたがって行動するときは，エージェントのプログラムから対応するスクリプトを起動する．一方，スクリプトからみると，エージェントはスクリプトの状態遷移規則から呼ばれる関数 (エージェント関数) を提供する (図 6.2)．

前述のようにエージェントごとにエージェント関数を用意することにより，大枠で



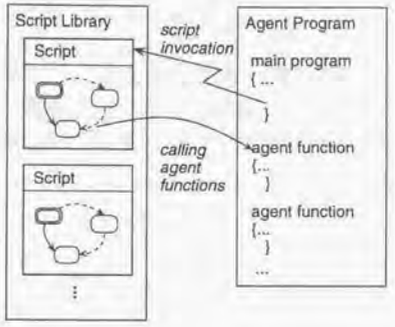



図 6.2: スクリプトとエージェントの関係

はスクリプトにしたがって動くものの，個々の振舞いはエージェント独自なものとすることができる．もし，対応するエージェント関数が定義されていない場合は同名のスクリプト関数が呼ばれる．すなわち，スクリプト関数がデフォールトの振舞いを定義していることになる．したがって，既存のスクリプトを継承した新たなスクリプトを定義し，その中でスクリプト関数だけを定義し直すことによって，スクリプトを段階的に定義することもできる．

### 6.3.3 状態遷移規則

状態遷移規則はその規則が起動する条件を定義した条件部と条件部が満足する時に実行される実行部からなる．条件部には (1) メッセージの到着に関する条件 (*msg-condition*)，(2) スクリプト変数に関する条件 (*script-var-condition*)，(3) 時間経過に関する条件 (*timeout-condition*) が記述できる．

ここで，メッセージはメッセージクラスのインスタンスとして表現される．メッセージクラスはインスタンスとなるメッセージのスロットを定義し，メッセージはスロットとその値の組として表現される．メッセージクラス間でスロット定義が継承される．

メッセージ条件はメッセージクラス (ないしは，そのサブクラス)，送り手，スロット値に関する条件を記述したメッセージパタンによって定義する．メッセージクラス

間にも継承関係があることにより，特定のクラスだけではなく，サブクラスのメッセージも条件に含めることが可能になる．

また，実行部には (1) 他の状態への遷移，(2) スクリプト変数の更新，(3) スクリプトの実行制御，(4) メッセージの送信，(5) スクリプト関数，エージェント関数の呼び出しをはじめとする手続きが記述できる．スクリプト実行制御に関しては後述する．

### 6.3.4 複数スクリプトの同時実行

スクリプトから他のスクリプトを起動することができる．ここで新たにスクリプトを起動したスクリプトを親スクリプトと呼び，新たに起動された方のスクリプトを子スクリプトと呼ぶ．スクリプトは基本的には並行的に実行される[2]．

一つのスクリプトを一つの協調プロセスに対応させ，複数のスクリプトを同時並行的に実行させることで，一つのエージェントにおける複数の協調プロセスを表現できる．さらに，ここでは起動元の親スクリプトのスクリプト変数を子スクリプトの状態遷移規則からアクセスできるようにしている．各協調プロセスに対応する子スクリプト間の通信を親スクリプトのスクリプト変数を媒介にして行うことで，協調プロセス間の競合解決メカニズムが記述できる．

エージェントはメッセージを受け取ったとき，現在実行中のスクリプト群の中から，条件部を満足させる状態遷移規則を探し出し，そのその実行部を実行する．このとき探索の順序は子スクリプトから親スクリプトへと行なわれる．すなわち，起動の若い順に探索が行われる．したがって，最初に起動されたスクリプトが最後に探索されることになる．すなわち，条件部が成立する状態遷移規則が複数のスクリプトに存在する場合は，子スクリプトの状態遷移規則を優先させるようにしている．これは，例えば子スクリプトで実行すべき状態遷移規則がない時に，親スクリプトがバックグラウンドで何らかの処理を行うことを想定している．

[2]起動元のスクリプトの実行が呼び出されたスクリプトの動作が終了するまで一時的に中断するモードもある．プログラミングの便宜上スクリプトをサブルーチンコールのように使うようにするためである．



```
(define-script script-name lambda-list {script-options}*)
script-options ::= :initial-state initial-state-name
                 | :script-vars ({script-var | (script-var initial-value)}*)
                 | :inherits-from script-name

(define-script-state (state-name script-name) [ :on-entry lisp-form ]
                     [ :rules ({rule}*) ])
rule ::= (:when condition :do action) | :inherited
condition ::= msg-condition | timeout-condition
            | [ msg-condition | timeout-condition ] script-var-condition
msg-condition ::= (msg message-class [ sender ] {slot-name value-pattern}*)
timeout-condition ::= (timeout lisp-form)
script-var-condition ::= (test lisp-form)
```

図 6.3: スクリプト定義のシンタックス (部分)

### 6.3.5 スクリプト記述のシンタックス

現バージョンの AgenTalk は Common Lisp 上に実装している．AgenTalk 自体は Common Lisp で定義したマクロないし関数群として実装され，AgenTalk のプログラム中から Common Lisp の関数が呼び出せる．

`define-script` というマクロでスクリプトの宣言を行い，`define-script-state` というマクロでスクリプトの各状態を定義している．図 6.3 にスクリプト定義のシンタックス (一部) を示す．`:on-entry` オプションにより，状態に遷移した時に実行される手続きを定義することができる．状態の定義の中に，その状態で有効な状態遷移規則を `:rules` オプションで定義している．なお，言語仕様の概要を本論文の付録としてつけている．

## 6.4 メタレベル制御

ここでは AgenTalk の拡張機能の一つとしてメタレベル制御の導入について述べる．具体的にはスクリプトの実行制御を他のスクリプトから行なうことを考える．す

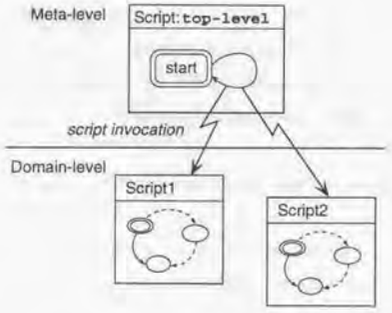



図 6.4: メタレベル制御

なわち，プロトコル記述と同様にスクリプトから他のスクリプトを起動し，その実行を制御できるようにする．メタレベル制御自身をスクリプトとして記述することにより，スクリプト記述における継承機能が活用できるという利点が生まれる．

具体的には，エージェントにエージェントの実行全体を管理する一種のトップレベルのスクリプトを設け，このスクリプトが必要に応じて，他のスクリプトを起動するようにする．このトップレベルのスクリプトは一種のメタレベルの制御を実現していると考えることができる (図 6.4)．

### 6.4.1 スクリプト実行制御のプリミティブ

前述のスクリプト起動のプリミティブ `run-script` はその返却値として実行コンテキスト (*context*) を返す．この実行コンテキストを元にスクリプトの実行の中止，一時停止，再開できるようにする (表 6.1)．

また，実行コンテキストからは現在の状態，親の実行コンテキスト，子の実行コンテキスト，及び子孫の実行コンテキストを求められるようにする (表 6.2)．スクリプトでは実行コンテキストをスクリプト変数に代入することによって，スクリプト変数に関する条件として状態遷移規則の条件部に記述できる．これにより，スクリプトの実行状態を基にスクリプトの実行を制御するようなスクリプトを記述することが可能になる．



表 6.1: スクリプト実行制御プリミティブ

| primitive | description |
|---|---|
| `run-script` *script-name-and-options* {*argument*}* | invokes a script |
| `exit-script` [ *return-value* ] [ *context* ] | terminates the script execution |
| `suspend-script` [ *context* ] | suspends the script execution |
| `resume-script` [ *context* ] | resumes the script execution |

表 6.2: スクリプト実行コンテキストへアクセスのプリミティブ

| primitive | description |
|---|---|
| `current-state` *context* | returns the current state name |
| `parent-context` *context* | returns a context of a immediate parent |
| `child-contexts` *context* | returns a list of child contexts |
| `decendant-contexts` *context* | returns a list of decendant contexts |

### 6.4.2 例外の通知

実際の応用システムにおいてはいろいろな例外的な状況が発生する．ここでは例外的な状況に対処するためのメカニズムを考える．まず，例外が発生したことを通知するためのメッセージクラスを設ける．基本のクラスとして `exception` というメッセージクラスを設け，種々の例外に対してこのクラスから派生したメッセージクラスを設ける．

例外的な状況はメッセージの配送エラーなどいろいろ考えられるが，ここでは予期しないメッセージが到着した場合を考えてみよう．ここで，実行中のスクリプトは現在エージェントが処理可能なメッセージに対応する状態遷移規則をもっているとの前提をおく．したがって，メッセージが到着したときにそのメッセージに対応する状態遷移規則が存在しない場合は例外 (エラー) となる．このエラーが発生したときは例外通知メッセージを自分自身に送ることによって，例外が発生したことをスクリプトが知ることができるようにする．

具体的にはトップレベルのスクリプトにおいて，すべてのメッセージに満足するよ

うなメッセージパタンを条件部に持つ状態遷移規則を書いておく．この規則の実行部には例外通知のメッセージ (exception クラスのサブクラス (message-unmatched) のメッセージ) を自分自身に送るようにしておく．トップレベルのスクリプトは一番最後に探索されるので，もし，他のスクリプトで処理される規則がない場合はこのトップレベルのスクリプトの状態遷移規則が実行されることになる．また，例外通知の送出はトップレベルのスクリプトで行なわれるため，個々のユーザの記述するプロトコルで記述する必要はない．ユーザが記述するスクリプトでは例外通知メッセージの処理を記述することになる．

### 6.4.3 プロトコルの動的変更

開環境下での協調ではどのプロトコルにしたがうかをあらかじめ厳密に決めることができなかったり，または状況に応じてプロトコルを動的に変更できることが望まれる．ここでは，そのような状況に対応するためにスクリプトの実行中に動的にスクリプトを変更することができるようにすることを考える．これにより，例えば，最初は基本的なプロトコルのスクリプトを起動しておいて，状況の変化に伴いより詳細なスクリプトに動的に切替えることが可能になる．

ここでは実行コンテキストを引数とする他のスクリプトに変更するためのプリミティブ (change-script) を導入する．AgenTalk においてはスクリプト記述に継承機能を導入しており，スクリプトの切替えにおいて継承関係を活用する．ここでは継承される方のスクリプトをスーパクラスのスクリプト，継承する方のスクリプトをサブクラスのスクリプトと呼ぶ[3]．スクリプトの変更においては以下の条件を満足することが必要となる．

- 共通のスーパクラスのスクリプトを持つこと．
- 状態の扱い: 現在の状態と同名の状態が存在する場合はそのまま移行する．同名の状態が存在しないときは変更できない．

スーパクラスからサブクラスのスクリプトを変更する場合は状態がそのまま継承されると考えられるから，問題なくスクリプトの動的な変更が可能となる．また，サブクラスからスーパクラスに変更する場合も状態がもとのスクリプトにも存在すれば変更できることになる．

[3]スクリプトの定義をクラスと見なし，起動したスクリプトをそのインスタンスとみなすと，オブジェクト指向言語においてクラスのインスタンスを実行時に変更すること (Common Lisp における change-class) に対応する．



```
find-scripts context [ message ]
change-script context new-script [ unprocessed-message ]
```

図 6.5: スクリプトの動的変更に用いられるプリミティブ

スクリプトを変更するのに伴って，スクリプト変数が少なくなる場合はそのスクリプト変数の値は一時的に待避される．将来，また，スクリプトの変更が発生して，一時的に待避されている値がある場合は元の値に戻される．そうでない場合は初期値に設定される．

### 6.4.4 スクリプト動的変更の契機

処理できないメッセージを受け取ったときにスクリプトの動的変更が試みられる．処理できないメッセージを受け取ったときは，トップレベルのスクリプトにおいてメッセージ未処理 (message-unmatched) の例外通知のメッセージが自分自身に送られる．このメッセージに対するデフォールトの処理はトップレベルのスクリプトに記述することができ，そこでは受け取ったメッセージを処理できるスクリプトを探し，そのスクリプトに現在の実行コンテキストのスクリプトを変更し，処理を継続することを行う[4]．

ここで変更先のスクリプトの候補を知るために find-scripts というプリミティブを用意する．これは，実行コンテキストと未処理のメッセージをオプションを引数にとり，現在の実行コンテキストから変更可能なスクリプトのリストを返す (図 6.5)．

メタレベル制御としては変更可能なスクリプトの候補を求め，その中から得られたスクリプトに実行コンテキストのスクリプトを変更することになる．スクリプトの変更のために change-script というプリミティブを導入する (図 6.5)．これはスクリプトの実行コンテキストと変更後のスクリプトを引数にとり，実行コンテキストのスクリプトを変更する．なお，change-script にオプショナル変数として，未処理のメッセージを渡すことができる．これは満足される状態遷移規則がないという状況で

[4]各スクリプトにおいてメッセージ未処理の例外通知の処理をカスタマイズすることも可能である．

スクリプトが変更される場合，スクリプトを変更するときに同時に未処理のメッセージの処理を行うためである．図 6.6 にトップレベルスクリプトの例を示す．

## 6.5 記述例

AgenTalk の機能を検証するために契約ネットプロトコルと AgenTalk の継承機能を活用した契約ネットプロトコルの拡張を記述する．また，前述のメタレベル制御の例についても述べる．

### 6.5.1 契約ネットプロトコルの記述

#### マネージャの記述

契約ネットプロトコルにおけるタスク割当は，基本的には次のように行われる．まず，実行すべきタスクを持っているエージェント (マネージャ) がタスク公告 (*Task Announcement*) メッセージを放送する．これに対して，タスクを請け負いたいエージェントが入札 (*Bid*) メッセージを出す．マネージャが入札の中から実際にタスクを割り当てるエージェント (コントラクタ) を決定し，落札 (*Award*) メッセージを出す．

このような契約ネットプロトコルにしたがうマネージャの振舞いを AgenTalk を用いて記述するには，まず，これらのメッセージの種類をメッセージクラスとして定義する．次にマネージャの状態遷移 (図 6.7) をもとにタスク割当のスクリプト (cnet-manager) を定義する．具体的なプログラム例を図 6.8 に示す．$ のマクロはスクリプト変数へのアクセス，また，! のマクロはエージェント関数の起動を表している[5]．このスクリプトは割り当てるべきタスクが発生した時を出発点としている．すなわち，タスクが発生した時にエージェントはこのスクリプトを起動することになる．

状態 *start* ではエージェント関数 announce-task を呼び出している．このエージェント関数はタスク公告を放送し，タスク割当の ID を返すという仕様になっている．

[5]ただし，! がマクロ名の位置に現れない場合 (図 6.8 中では !($ contract-id)) は，! に続く lisp 式 (この場合スクリプト変数 contract-id へのアクセス) をパターンマッチ時に評価することを表している．詳しくは付録を参照されたい．



```
;;;script declaration
(define-script top-level ()
  :initial-state 'start)
;;;state transition rules
(define-script-state (start top-level)
  :rules
   ;;this rule matches a message-unmatched message.
  ((:when (msg message-unmatched :message ?msg)
    ;;find a script which can handle the message (?msg).
    ;; context refers to the current execution context.
    ;; note that Common Lisp functions/macros can be used here.
    :do (let ((context-list (descendant-contexts context)))
          ;;try each context in the descendant contexts.
          (dolist (this-context context-list)
            ;;is an appropriate script found?
            (let ((scripts (find-script this-context ?msg)))
              (when scripts
                ;;if found, change the executing script of this context
                ;; to the top of the script list.
                (change-script this-context (car scripts) ?msg)
                ;;exit the dolist.
                (return nil))))))
   ;;other default rules such as handling a message-not-delivered
   ;;  message should come here...
   ...
   ;;the last rule matches any message.
   (:when (msg message)
    ;;send an exception notifying message to itself (self).
    :do (send-message self 'message-unmatched :message msg))
  ))
```

図 6.6: トップレベルスクリプトの例 (一部)

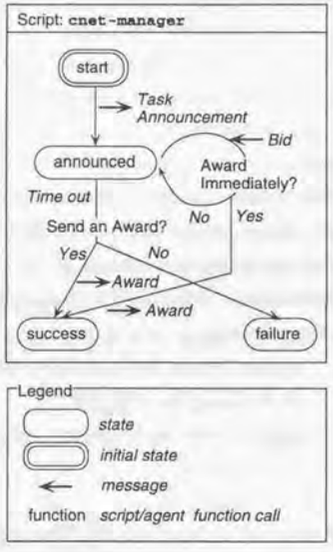



図 6.7: 契約ネットプロトコルにおけるマネージャの状態遷移



```
(define-script cnet-manager (task)
  :initial-state 'start
  :script-vars (bid-queue contract-id))
;;;start state
(define-script-state (start cnet-manager)
  ;;broadcast a task-announcement message by invoking the agent function announce-task.
  :on-entry (progn (setf ($ contract-id) (! announce-task))
                   (goto-state 'announced)))
;;;announced state
(define-script-state (announced cnet-manager)
  :rules
   ;;wait for a bid message with the same contract-id.
  ((:when (msg bid :contract-id !($ contract-id))
    :do (if (! send-award-immediately-if-possible msg)
            (goto-state 'success)
            (push msg ($ bid-queue))))
    ;;check a timeout condition.
   (:when (timeout (task-expiration task))
    :do (if (! send-award-if-possible)
            (goto-state 'success)
            (goto-state 'failure)))))
;;;success state
(define-script-state (success cnet-manager)
  :on-entry (exit-script t))
;;;failure state
(define-script-state (failure cnet-manager)
  :on-entry (exit-script nil))
```

図 6.8: 契約ネットプロトコルにおけるマネージャのスクリプト定義

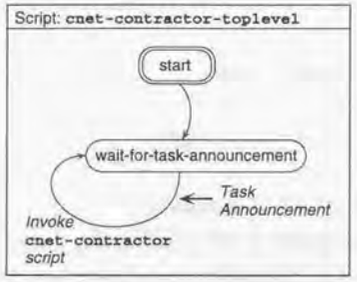



図 6.9: 契約ネットプロトコルにおけるコントラクタのトップレベル

したがって，エージェントごとに具体的なタスク公告を定義できる．状態 *announced* では入札のメッセージを待つ状態遷移規則と，時間切れを監視する状態遷移規則の二つが有効になっている．また，この例では状態 *success*, *failure* に遷移した時は単にスクリプトの実行を終了する (`exit-script`) ようになっている．

### コントラクタの記述

タスク公告に対して，入札を出すコントラクタの振舞いはどうなるだろうか？もし，対象になっているタスクが自分のところで実行できる場合は，そのまま判断して返答すればよいが，もし，自分のところだけでは実行できない場合は，タスクを複数のサブタスクに分解し，個々のサブタスクを他のエージェントに割り当てることも考えられる．この場合，生成されたサブタスクごとにタスク割当のスクリプトを起動することになる．

このような振舞いはスクリプトの中から他のスクリプトを起動する機能を用いることによって記述できる．例えば一種のトップレベルのスクリプトでタスク公告を監視していて，適当なタスク公告が来た場合に，コントラクタのスクリプト (`cnet-contractor`) を起動する (図 6.9)．`cnet-contractor` のスクリプトでは，例えば，サブタスクへの分解が必要か調べ，もし，必要であれば各サブタスクに関して `cnet-manager` のスクリプトを起動して，サブタスクの割当を行う．この場合，すべてのサブタスクの割当ができた段階 (*success* の状態) でもとのタスク公告に対して入



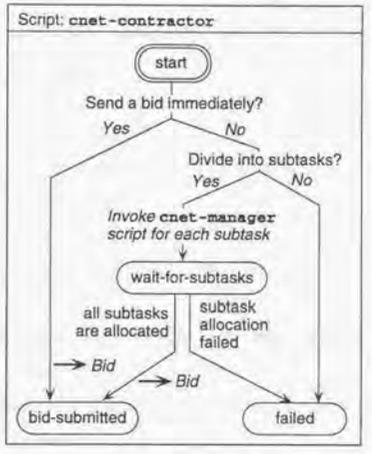



図 6.10: 契約ネットプロトコルにおけるコントラクタの状態遷移 (例)

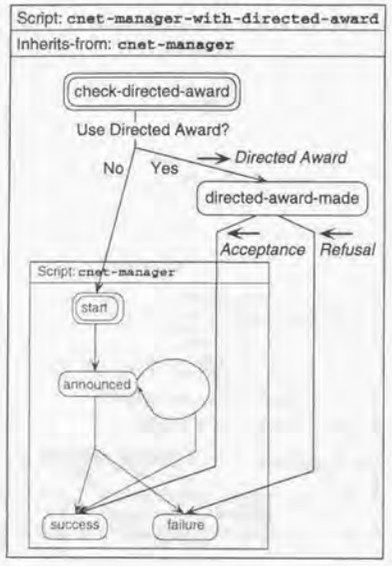



図 6.11: 指定落札を導入した契約ネットプロトコルの拡張

札する (図 6.10).

### 6.5.2 契約ネットプロトコルの拡張

契約ネットプロトコルの拡張として，タスクを割り当てるべきエージェントがあらかじめわかっている場合の指定落札 (*Directed Award*) メッセージが定義されている [55]．この場合は指定落札を受け取ったエージェントは受理 (*Acceptance*) か拒否 (*Refusal*) の返答を送ることなる．この拡張を導入したときのマネージャの状態遷移は図 6.11 のようになる．この図に示すように `cnet-manager-with-directed-award` のスクリプトは基本的な契約ネットプロトコルのマネジャのスクリプト `cnet-manager` をもとに定義できる．また，このプログラム記述例を図 6.12 に示す．



```
(define-script cnet-manager-with-directed-award (task contractor)
  :initial-state 'check-directed-award
  :inherits-from cnet-manager)
(define-script-state (check-directed-award cnet-manager-with-directed-award)
  :on-entry (if (! directed-award-made-p)
                (goto-state 'directed-award-made)
                (goto-state 'start)))

(define-script-state (directed-award-made cnet-manager-with-directed-award)
  :rules
  ((:when (msg acceptance :contract-id !($ contract-id))
    :do (goto-state 'success))
   (:when (msg refusal :contract-id !($ contract-id))
    :do (goto-state 'failure))
   ))
```

図 6.12: 指定落札を導入した場合のスクリプト定義

### 6.5.3 マルチステージネゴシエーションの記述

第 3,4 章で述べたマルチステージネゴシエーションは契約ネットプロトコルを一般化したものと考えることができる [6]. 契約ネットプロトコルでは基本的にはタスク公告，入札，落札の過程でタスクの割当が決定される．それに対して，マルチステージネゴシエーションでは局所的な仮の割当の影響を繰り返し交換することを許す．そこでは大域的な制約を満足させるためにエージェント間で交換すべき情報を定義しており，割当が最終的に決まるまで，何度も仮の割当がやり直される場合もある．

マルチステージネゴシエーションにおけるこのような割当の再試行を契約ネットプロトコルに導入することを考えよう．そのためには，まず，タスク公告，入札，落札を取り消す取消 (*Cancel*) メッセージ，実際に割当を決定するコミット (*Commit*) メッセージ，およびその確認を行うコミット終了 (*Committed*) メッセージのメッセージクラスを追加する．この時，マネージャの状態遷移は図 6.13 に示すように拡張される[6]．契約ネットプロトコルにおいて割当が成功したという段階 (*success* の状態) は，仮の割当となる．仮の割当が実現できた後，落札を送ったエージェントに対し，コミットメッセージを送り，割当をコミットすることを依頼する．コミットメッセージを送ったエージェントから，コミット終了メッセージが帰ってきた段階で割当が最終的に決定することになる．また，ここでは取消メッセージを受け取った時は，単に *failure* の状態に移行することにしている．

一度割当に失敗した時，繰り返して割当を試みる場合は，ここで述べたスクリプト (`msn-manager`) を呼び出す親スクリプトにおいて再び割当を試みるべきか決定する．このように割当の再試行に関する部分を別のスクリプトとして独立に記述できる．また，同様にコントラクタに関してもタスク公告自体が取消になる可能性があるので，取消メッセージが来るかチェックし，取消があった場合はサブタスクのコントラクタ (の候補) に対して取消メッセージを送るように拡張することになる．

また，マルチステージネゴシエーションでは一つのエージェントに複数のネゴシエーションの過程が存在し，それらがエージェント内の共通の資源を使用することによって競合関係になる場合がある．トップレベルのスクリプトのスクリプト変数を用いて共通の資源の使用状況を記述し，個々のネゴシエーションの過程を記述している

[6] オリジナルのマルチステージネゴシエーションでは指定落札に相当するもののみを対象としていたが，ここではより一般化した場合を想定している．



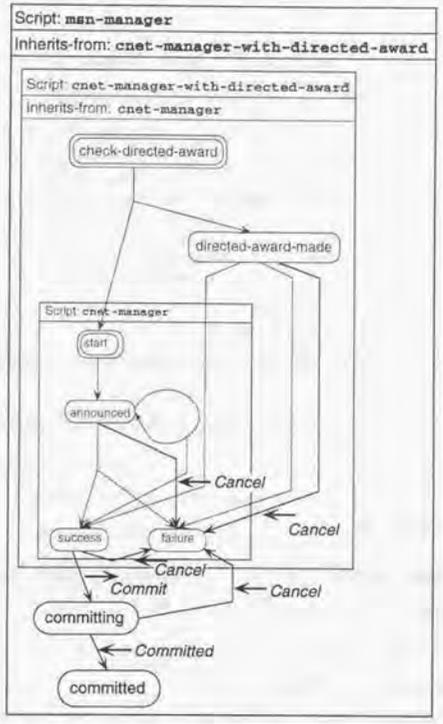



図 6.13: タスク割当における再試行の導入

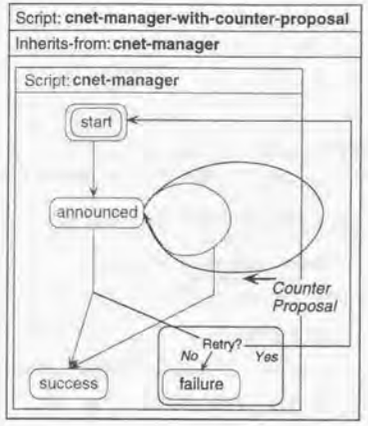



図 6.14: 逆提案を導入した契約ネットプロトコルの拡張

子スクリプトから参照，更新することにより，共通資源の使用に関する競合解決が表現できる．

### 6.5.4 メタレベル制御の例

次にメタレベル制御，具体的にはスクリプトの動的変更機能を契約ネットプロトコルを例題として説明する．基本的な契約ネットプロトコルではマネージャは原則としてタスク公告に対して入札を待つことになる．文献 [55] にあるように入札に対して即時返答 (immediate response) が仮に送られてきたとしよう．これは入札が行えないエージェントがその理由 (忙しい，入札資格無し，タスクの優先度が低いなど) を返答するものである．また，タスク公告に対して逆提案 (*Counter Proposal*) を出してくることも考えられる．基本的なプロトコルでは逆提案に対してマッチする状態遷移規則が存在せず，みすみす有益な情報を失ってしまうことになる．

逆提案を採り入れた契約ネットプロトコルの拡張では状態 *announced* の状態において入札に加えて逆提案を処理できるようにする必要がある．例えば，受けとった逆提案を保存しておいて，期限までに適当な入札が来ず，落札が行なえなかったとき，こ



```
(define-script cnet-manager-with-counter-proposal (task)
  :script-vars (counter-proposal-list)
  :inherits-from cnet-manager)
;;;redefine the announced state.
(define-script-state (announced cnet-manager-with-counter-proposal)
  :rules
  (;;state transition rules are inherited.
   :inherited
   ;;a state transition rule which handles a counter-proposal message is added.
   (:when (msg counter-proposal :contract-id !($ contract-id))
    ;;record counter proposals.
    :do (push msg ($ counter-proposal-list)))))
;;;redefine the failure state.
(define-script-state (failure cnet-manager-with-counter-proposal)
  ;;check counter proposals received so far.
  :on-entry (if (! retry-using-counter-proposal-p)
                ;;go back to the start state.
                (goto-state 'start)
                ;;exit the execution of this script.
                (exit-script nil)))
```

図 6.15: 逆提案を導入した場合のスクリプト定義

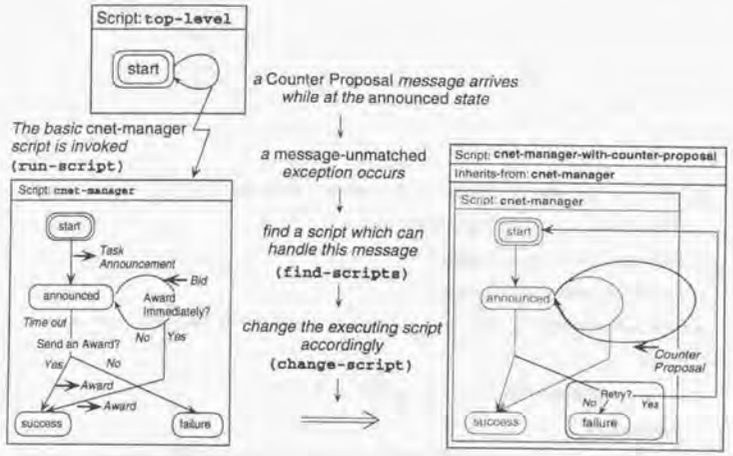



図 6.16: スクリプトの動的変更例

れまでに受けとった逆提案を調べ，内容を変更したタスク公告を再び放送することが考えられる．このように拡張を行なったときのマネージャの状態遷移図を図 6.14 に，また，プログラム例を図 6.15 に示す．

もちろん，最初から拡張されたプロトコルを記述したスクリプトを起動することも考えられる．しかし，いろいろな拡張も考えられ，最初からどの拡張を使ったらよいかはわからないとすれば，まず，最初は基本の契約プロトコルのスクリプトを起動し，実行スクリプトの変更機能を活用すればよいことになる．この場合，基本の契約ネットプロトコルにおいて `counter-proposal` のメッセージが到着したとき，`message-unmatched` の例外通知が行なわれる．例外通知に処理として `find-scripts` を用いて処理できないメッセージに対応する状態遷移規則を含むスクリプトを見つける．これによって，逆提案の拡張をした契約ネットプロトコルのスクリプトが見つかり，このスクリプトに実行コンテキストのスクリプトが変更される (図 6.16)．

このような機能を用いることにより，最初から特定のプロトコルを決めることなく，基本的なプロトコルから始めて，必要に応じてプロトコルを変更することができる．例えばエージェントの間でどのプロトコルを使うか合意をとるときには基本的なプロトコルのみについて合意をとり，後から必要に応じて，自動的に特定の拡張プロトコルに変更するということが実現できる．



## 6.6 議論

**プロトコル記述における継承機能:** ここで述べたスクリプトの段階的な記述の例は状態を新たに追加するものであった．段階的なプロトコル記述として，その他に状態を詳細化することが考えられる．この場合，継承元のスクリプトの状態遷移規則のうち，詳細化された状態へ遷移するものについては，単純な継承機能だけでは，再定義が必要になる [37]．並行オブジェクト指向言語においても，同様に継承を用いたプログラムの再利用がうまくできない場合があることが指摘されており，いくつかの解決手法が提案されている [46]．スクリプトの再利用を促進するようにスクリプトの継承機構を拡張することは今後の課題である．

また，一般のオブジェクト指向言語における継承では，多重継承が議論の対象となる．有限状態機械 (スクリプト) の継承において多重継承を導入することは，複数の異なる有限状態機械を継承して新たな有限状態機械を定義することになる．継承した複数の有限状態機械をどのように組み合わせて新たな有限状態機械とするべきかは検討の余地が残っており，現状では単一継承としている．スクリプトの継承機能に多重継承をいかに導入するかも今後の課題として残っている．

**分散探索との関係:** マルチエージェントにおける問題解決の多くは分散探索の枠組で考えることができる [39]．分散探索のアルゴリズムはエージェント間のメッセージの交換の形でとらえられるものも多い (例えば [24, 64])．これらのアルゴリズムをスクリプトの形で記述することにより，種々の領域への応用を容易にすることも考えられる．

**メタレベル制御:** 本章で述べたメタレベル制御では，処理不能のメッセージが到着したとき，スクリプトを動的に切替えるという方式を提案した．エージェントが複数の協調プロセスにかかわっている場合に処理不能となったメッセージがどの協調プロセ

スに属するかは，単純には識別できない．実際にはメッセージクラスに協調プロセスを識別するための識別子を格納するスロットを設けるなどのプログラミング上の工夫が必要となる．これらを言語機能としてサポートするのは今後の課題として残っている．

**エージェントプラニング** スクリプトはプロトコルにおけるエージェントの状態遷移を表現したものである．したがって，エージェントの動作 (プラン) を表現することも可能である．特に動的にスクリプトを修正するような機構を導入することによって，エージェントのプラニングにも対応することが可能となろう．これによりスクリプトを用いて協調プロトコル記述とプラニング記述を一つの枠組のなかで行なうことができるようになる．スクリプトの動的変更も今後の課題として残っている．

## 6.7 まとめ

本章では，マルチエージェントにおける協調プロトコルを記述する言語 AgenTalk を提案した．AgenTalk の設計方針を述べるとともに契約ネットプロトコルならび，拡張された契約ネットプロトコルを記述することにより，AgenTalk の記述能力の検証を行った．

AgenTalk では

- オブジェクト指向言語に見られる継承機能を導入した段階的なプロトコル記述
- エージェントが独自にプロトコルをカスタマイズするためのインタフェース

を実現している．これにより，種々のプロトコルをライブラリとして蓄積し，アプリケーションに依存したプロトコルを記述しやすくしている．

# 第 7 章

# 結論

本論文ではマルチエージェントシステムにおける協調メカニズムとしてマルチステージネゴシエーション，市場モデルに基づく均衡的アプローチによる分散資源割当を提案し，さらにこのような協調メカニズムを実装するための協調プロトコル記述言語 AgenTalk を提案した．以下にその成果を要約し，今後の課題を述べる．

第 1 章で本研究の背景を概説するとともに本研究の目的を明確にした．さらに第 2 章でマルチエージェントシステムにおける協調メカニズム研究状況の概要を述べ，本論文の位置付けを明確にした．

第 3 章では協調プロトコルの一例としてマルチステージネゴシエーションを取り上げ，そこで扱っている問題の定式化を行なった．具体的にはシステム全体の目標 (グローバルゴール) と個々のエージェントの目標 (サブゴール) を考慮に入れた探索空間を定式化した．さらに定式化した探索空間に基づいてエージェント間で交換すべき情報を明らかにし，複数のグローバルゴール間の競合を検出する手法を提案した．続く第 4 章ではマルチステージネゴシエーションにおける探索戦略 (3 フェーズプロトコル) の定量的な評価を行ない，制約が弱い時により効果が大きいことを明らかにした．

第 5 章では市場モデルに基づいて資源割当を行なう “均衡的アプローチ” を提案し，シミュレーション実験を通して市場モデルの適用可能性を示した．ここではメッセージの交換は資源の売り手と買い手の間に限定されている．さらにエージェント間で交換される情報は市場モデルに基づき価格情報の通知 (売り手から買い手へ) と資源割当要求 (買い手から売り手へ) に限定されている．マルチステージネゴシエーションでは資源の割当が行なうか行なわないかのゼロ一の問題であったのに対し，ここでの市

場モデルの応用では割り当てる資源の量を扱えるようになっている．通信ネットワークを題材として例題を取り上げ，シミュレーション実験により提案手法の性質を調べた．資源の使用率の均等化を目標にした実験の結果，各資源の使用率の分散に振動を起こすことがわかったが，感度係数を小さくしたり，価格情報の遅延に対して推定手法を導入することにより，振動の度合を小さくできることを示した．

さらに，第6章では協調プロトコルの記述言語 AgenTalk を提案した．AgenTalk ではプロトコル定義に継承機構を導入し，プロトコルの段階的定義を実現した．さらに応用領域にカスタマイズするためのインタフェースを設けているのが特徴である．契約ネットプロトコルならびにそれを拡張したプロトコルを AgenTalk を用いて記述することにより，プロトコルの段階的定義機能を実証した．さらに AgenTalk で提案した枠組により協調プロトコルのメタレベル制御を実現できることを示した．

本論文では，探索空間のモデル (マルチステージネゴシエーション) やエージェントの組織構造 (市場モデル) に基づいてエージェント間で交換すべき情報を規定した．ここでは，問題の構造，エージェント組織の構造が求解の途中で変更することがない静的な場合をとりあげていた．しかし，実世界のアプリケーションではゴールが途中で新たに追加されたり，または消滅したり，問題の探索空間が途中で変化することが考えられる．さらには，市場モデルにおける買い手，売り手が増加，減少するようにエージェントの組織構造が変化することも考えられる．このような動的な変化に対処できるように本論文で提案した協調メカニズムを拡張していくことは今後の課題として残っている．これについては，協調プロトコル記述言語 AgenTalk で提案したメタレベル制御の考え方を導入して，種々の問題の変化に対応することが解決手法の一つと考えており，AgenTalk を用いて種々の協調メカニズムを記述実験をすすめていきたいと考えている．

現在，協調メカニズムの研究は，個々の事例を題材に研究している場合が多く，統一的なモデルなり理論は，これから作られていくという段階にあるといえる．そこでは本論文で述べた市場モデルの応用にも見られるように単に計算機科学の中だけにとどまらず，社会科学などとの接点も見られる．今後，種々の分野との連携から新しいアイデアも生まれてこよう．

また，工学的な見地からみれば，高性能の計算機が安価に入手できるようになり，さらに高速なネットワークが普及しつつある今の状況は，協調メカニズムが単なる机



上の話にとどまらず，実際のシステムの構築に応用していく絶好の機会であるともいえる．実システムへの応用と，理論・モデルの構築がうまく互いにかみあっていくように今後も研究をすすめていきたいと考えている．

# 謝辞

本論文をまとめるにあたり東京大学工学部電子情報工学科 石塚満教授には終始懇切丁寧な御指導，御教示を賜わりました．ここに謹んで感謝の意を表します．

また，内容について多くの御指導，御助言を賜わりました東京大学工学部電子情報工学科 斉藤忠夫教授，東京大学工学部電気工学科 田中英彦教授に深く感謝致します．東京大学工学部電子情報工学科 近山隆助教授，東京大学生産技術研究所 喜連川優助教授，東京大学工学部電子情報工学科 相田仁助教授には多くの有益な御教示を賜わりました．厚く御礼申し上げます．

本研究は主に NTT 情報通信処理研究所，コミュニケーション科学研究所にて多くの方々の御指導を得て行なわれました．研究の機会を与えて頂くともに，御指導頂いた故 村上国男博士 (元 NTT 情報通信処理研究所知識処理研究部長)，大阪大学基礎工学部情報工学科 西川清史教授 (元 NTT コミュニケーション科学研究所長)，同志社大学工学部知識工学科 河岡司教授 (前 NTT コミュニーケション科学研究所長)，NTT コミュニケーション科学研究所 松田晃一所長，中野良平グループリーダ，大里延康グループリーダに厚く御礼申し上げます．

また，京都大学工学部情報工学教室 石田亨教授 (元 NTT コミュニケーション科学研究所 主幹研究員) には本研究を進めるにあたり終始丁寧な御指導，御教授を賜わりました．ここに深く感謝致します．

マルチステージネゴシエーションの研究に関しまして University of Massachusetss at Amherst において熱心な御指導を頂きました Victor R. Lesser 教授に感謝致します．また，Clarkson University の Susan E. Conry 教授，Robert A. Meyer 教授には有益な御討論を頂きました．ここに感謝致します．

市場モデルの応用の研究に関して NTT コミュニケーション科学研究所に招聘教授として滞在された University of California, Irvine の須田達也教授に多くの御指導，

御助言を頂きました．ここに感謝致します．

また，多くの御討論を頂いたNTTコミュニケーション科学研究所の赤埴淳一氏，横尾真氏，榛葉之氏，西部喜康氏(現NTT情報通信研究所)，松原繁夫氏，上野磯生氏，吉田仙氏をはじめ，皆様に感謝致します．また，NTT光ネットワークシステム研究所の藤井伸朗氏，依田育生氏には本研究で使用したネットワーク構成管理データベースを提供して頂きました．ここに感謝致します．また，協調プロトコル記述言語AgenTalkの言語仕様検討ならびに実装に御助力頂いた京都大学工学部情報工学教室石田研究室の皆様に感謝致します．

本論文をまとめるにあたってはNTT研究開発推進部研究広報・渉外部門山田豊通部門長，山内規義担当部長(現NTTマルチメディアネットワーク研究所)をはじめとする皆様のあたたかい御理解を頂きました．ここに記して深謝致します．

# 参考文献

[1] Barbuceanu, M. and Fox, M. S.: COOL: A Language for Describing Coordination in Multi Agent Systems, *Proc. First International Conference on Multi-Agent Systems (ICMAS '95)*, pp. 17–24 (1995).

[2] Bond, A. H. and Gasser, L. eds.: *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann (1988).

[3] Clearwater, S. H. ed.: *Market-Based Control: A Paradigm for Distributed Resource Allocation*, World Scientific Publishing (1996).

[4] Cohen, P. R.: Communicative Actions for Artificial Agents, *Proc. First International Conference on Multi-Agent Systems (ICMAS '95)*, pp. 65–72 (1995).

[5] Collin, Z., Dechter, R., and Katz, S.: On the Feasibility of Distributed Constraint Satisfaction, *IJCAI-91*, pp. 318–324 (1991).

[6] Conry, S. E., Kuwabara, K., Lesser, V. R., and Meyer, R. A.: Multistage Negotiation for Distributed Constraint Satisfaction, *IEEE Trans. Systems, Man and Cybernetics*, Vol. 21, No. 6, pp. 1462–1477 (1991).

[7] Conry, S. E., Meyer, R. A., and Lesser, V. R.: Multistage Negotiation in Distributed Planning, in Bond, A. H. and Gasser, L. eds., *Readings in Distributed Artificial Intelligence*, pp. 367–384, Morgan Kaufmann (1988).

[8] Conry, S. E., Meyer, R. A., and Pope, R. P.: Mechanisms for Assessing Nonlocal Impact of Local Decisions in Distributed Planning, in Gasser, L. and Huhns, M. N. eds., *Distributed Artificial Intelligence, Volume II*, pp. 245–258, Morgan Kaufmann (1989).

[9] Davis, R. and Smith, R. G.: Negotiation as a Metaphor for Distributed Problem Solving, *Artificial Intelligence*, Vol. 20, No. 1, pp. 63–109 (1983).

[10] Durfee, E. H. and Lesser, V. R.: Using Partial Global Plans to Coordinate Distributed Problem Solvers, *IJCAI-87*, pp. 875–883 (1987).

[11] Durfee, E. H., Lesser, V. R., and Corkill, D. D.: Distributed Problem Solving, in Shapiro, S. C. ed., *Encyclopedia of Artificial Intelligence, 2nd Ed.*, pp. 379–388, John Wiley & Sons (1992).

[12] Ferguson, D., Yemini, Y., and Nikolaou, C.: Microeconomic Algorithms for Load Balancing in Distributed Computer Systems, *8th International Conference on Distributed Computing System*, pp. 491–499 (1988).

[13] Finin, T., Fritzson, R., McKay, D., and McEntire, R.: KQML – A Language and Protocol for Knowledge and Information Exchange, in Fuchi, K. and Yokoi, T. eds., *Knowledge Building and Knowledge Sharing*, pp. 249–259, Ohmsha, Ltd. and IOS Press (1994).

[14] Haugeneder, H., Steiner, D., and McCabe, F. G.: IMAGINE: A Framework for Building Multi-Agent Systems, *Proc. Second International Working Conference on Cooperating Knowledge Based Systems (CKBS '94)*, pp. 31–64 (1994).

[15] Hogg, T. and Huberman, B. A.: Controlling Chaos in Distributed Systems, *IEEE Trans. Systems, Man and Cybernetics*, Vol. 21, No. 6, pp. 1325–1332 (1991).

[16] Holzmann, G. J.: *Design and Validation of Computer Protocols*, Prentice-Hall (1991).

[17] Huberman, B. A. and Hogg, T.: The Behavior of Computational Ecologies, in Huberman, B. A. ed., *The Ecology of Computation*, pp. 77–115, Elsevier Science Publishers (1988).

[18] Huhns, M. N. and Bridgeland, D. M.: Multiagent Truth Maintenance, *IEEE Trans. Systems, Man and Cybernetics*, Vol. 21, No. 6, pp. 1437–1445 (1991).



[19] Ishida, T.: Bridging Humans via Agent Networks, *Proc. 13th International Workshop on Distributed Artificial Intelligence* (1994).

[20] 石田亨, 桑原和宏: 分散人工知能 (1): 協調問題解決, 人工知能学会誌, Vol. 7, No. 6, pp. 945–954 (1992).

[21] 石田亨, 桑原和宏: 分散人工知能, 情報処理学会 (編), 情報処理ハンドブック, 13 編 5.1, pp. 1466–1468, オーム社 (1995).

[22] 梶原史雄, 桑原和宏, 石田亨: エージェントネットワーク Socia の AgenTalk による実装, 情報処理学会第 50 回全国大会, pp. 2–163–164 (1995).

[23] 亀田恒彦, 山下雅史: 分散アルゴリズム, 近代科学社 (1994).

[24] 北村泰彦, 辰巳昭治, 奥本隆昭: 分散問題解決のための波及型探索法とその評価, 情報処理学会論文誌, Vol. 35, No. 12, pp. 2651–2663 (1994).

[25] Kolb, M.: A Cooperation Language, *Proc. First International Conference on Multi-Agent Systems (ICMAS '95)*, pp. 233–238 (1995).

[26] 桑原和宏: マルチステージネゴシエーションによる分散資源割当, 1990 年代の分散処理シンポジウム, pp. 87–96, 情報処理学会 (1990).

[27] Kuwabara, K.: *AgenTalk Reference Manual (preliminary draft)* (1995).

[28] Kuwabara, K.: Meta-Level Control of Coordination Protocols, *Proc. Second International Conference on Multi-Agent Systems (ICMAS '96)*, pp. 165–172 (1996).

[29] Kuwabara, K. and Ishida, T.: Equilibratory Approach to Distributed Resource Allocation: Toward Coordinated Balancing, in Castelfranchi, C. and Werner, E. eds., *Artificial Social Systems, MAAMAW '92*, Lecture Notes in AI 830, pp. 133–146, Springer-Verlag (1994).

[30] Kuwabara, K., Ishida, T., and Nishibe, Y.: Market-Based Distributed Resource Allocation: Equilibratory Approach, *IJCAI-93 Workshop on Artificial Economics* (1993).

[31] Kuwabara, K., Ishida, T., and Osato, N.: AgenTalk: Coordination Protocol Description for Multiagent Systems, *Proc. First International Conference on Multi-Agent Systems (ICMAS '95)*, p. 455 (1995).

[32] Kuwabara, K., Ishida, T., and Osato, N.: AgenTalk: Describing Multiagent Coordination Protocols with Inheritance, *Proc. 7th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '95)*, pp. 460–465 (1995).

[33] 桑原和宏, 石田亨, 大里延康: AgenTalk: マルチエージェントシステムにおける協調プロトコル記述, 信学技報 AI 94-56 (1995).

[34] 桑原和宏, 石田亨, 大里延康: AgenTalk: マルチエージェントシステムにおける協調プロトコル記述, 電子情報通信学会論文誌 B-I, Vol. 79, No. 5, pp. 346–354 (1996).

[35] Kuwabara, K. and Lesser, V. R.: Extended Protocol for Multistage Negotiation, *Proc. 9th Workshop on Distributed Artificial Intelligence*, pp. 129–161 (1989).

[36] 桑原和宏, Lesser, V. R.: マルチステージネゴシエーションにおけるゴール間競合の検出, 情報処理学会論文誌, Vol. 32, No. 10, pp. 1269–1280 (1991).

[37] 桑原和宏, 篠原拓嗣, 大里延康, 石田亨: 協調プロトコル記述言語 AgenTalk の実現, 信学技報 AI 95-18 (1995).

[38] Labrou, Y. and Finin, T.: A Semantics Approach for KQML – A General Purpose Communication Language for Software Agents, *Proc. Third International Conference on Information and Knowledge Management (CIKM '94)* (1994).

[39] Lesser, V. R.: A Retrospective View of FA/C Distributed Problem Solving, *IEEE Trans. Systems, Man and Cybernetics*, Vol. 21, No. 6, pp. 1347–1362 (1991).

[40] Lesser, V. R. and Corkill, D. D.: Functionally Accurate, Cooperative Distributed Systems, *IEEE Trans. Systems, Man and Cybernetics*, Vol. 11, No. 1, pp. 81–96 (1981).



[41] Lesser, V. R. and Corkill, D. D.: The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks, *AI Magazine*, Vol. 4, No. 3, pp. 15–33 (1983).

[42] Lesser, V. R. and Erman, L. D.: Distributed Interpretation: A Model and Experiment, *IEEE Trans. Comput.*, Vol. 29, No. 12, pp. 1144–1163 (1980).

[43] Mackworth, A. K.: Constraint Satisfaction, in Shapiro, S. C. ed., *Encyclopedia of Artificial Intelligence*, pp. 205–211, John Wiley & Sons (1987).

[44] Malone, T. W., Fikes, R. E., Grant, K. R., and Howard, M. T.: Enterprise: A Market-like Task Scheduler for Distributed Computing Environments, in Huberman, B. A. ed., *The Ecology of Computation*, pp. 177–205, Elsevier Science Publishers (1988).

[45] Mason, C. L. and Johnson, R. R.: DATMS: A Framework for Distributed Assumption Based Reasoning, in Gasser, L. and Huhns, M. N. eds., *Distributed Artificial Intelligence, Volume II*, pp. 293–317, Morgan Kaufmann (1989).

[46] Matsuoka, S. and Yonezawa, A.: Analysis of Inheritance Anomaly in Object-Oriented Concurrent Programming Languages, in Agha, G., Wegner, P., and Yonezawa, A. eds., *Research Directions in Concurrent Object-Oriented Programming*, pp. 107–150, MIT Press (1993).

[47] Miller, M. S. and Drexler, K. E.: Markets and Computation: Agoric Open Systems, in Huberman, B. A. ed., *The Ecology of Computation*, pp. 133–176, Elsevier Science Publishers (1988).

[48] Nishibe, Y., Kuwabara, K., and Ishida, T.: Effects of Heuristics in Distributed Constraint Satisfaction: Towards Satisficing Algorithms, *Proc. 11th International Workshop on Distributed Artificial Intelligence*, pp. 285–302 (1992).

[49] Nishibe, Y., Kuwabara, K., Suda, T., and Ishida, T.: Distributed Channel Allocation in ATM Networks, *Proc. GLOBECOM '93*, pp. 417 – 423 (1993).

[50] 西田豊明: 協調型アーキテクチャによる知識の共有と再利用, 人工知能学会誌, Vol. 9, No. 1, pp. 23–28 (1994).

[51] Patil, R. S., Fikes, R. E., Patel-Schneider, P. F., Mckay, D., Finin, T., Gruber, T., and Neches, R.: The DARPA Knowledge Sharing Effort: Progress Report, in Nebel, B., Rich, C., and Swartout, W. eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, Morgan Kaufmann (1992).

[52] Rosenschein, J. S. and Genesereth, M. R.: Deals Among Rational Agents, *IJCAI-85*, pp. 91–99 (1985).

[53] Sandholm, T.: An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations, *Proc. 11th National Conference on Artificial Intelligence (AAAI '93)*, pp. 256–262 (1993).

[54] Sen, S. and Durfee, E. H.: The Role of Commitment in Cooperative Negotiation, *International Journal of Intelligent and Cooperative Information Systems*, Vol. 3, No. 1, pp. 67–81 (1994).

[55] Smith, R. G.: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver, *IEEE Trans. Comput.*, Vol. 29, No. 12, pp. 1104–1113 (1980).

[56] Smith, R. G. and Davis, R.: Frameworks for Cooperation in Distributed Problem Solving, *IEEE Trans. Systems, Man and Cybernetics*, Vol. 11, No. 1, pp. 61–70 (1981).

[57] Steiner, D., Burt, A., Kolb, M., and Lerin, C.: The Conceptual Framework of MAI$^2$L, in Castelfranchi, C. and Müller, J.-P. eds., *From Reaction to Cognition, MAAMAW '93*, Lecture Notes in AI 957, pp. 217–230, Springer-Verlag (1995).

[58] Vickrey, W.: Counter Speculation, Auctions, and Competitive Sealed Tenders, *Journal of Finance*, Vol. 16, pp. 8–37 (1961).



[59] Waldspurger, C. A., Hogg, T., Huberman, B. A., Kephart, J. O., and Stornetta, W. S.: Spawn: A Distributed Computational Economy, *IEEE Trans. Softw. Eng.*, Vol. 18, No. 2, pp. 103–117 (1992).

[60] Wellman, M. P.: A Market-Oriented Programming Environment and its Application to Distributed Multicommodity Flow Problems, *Journal of Artificial Intelligence Research*, Vol. 1, pp. 1–23 (1993).

[61] 山口治男, 藤井伸朗, 山中康史, 依田育生: ネットワーク構成管理データベース, *NTT R & D*, Vol. 38, No. 12, pp. 1509–1518 (1989).

[62] 依田育生, 山中康史, 藤井伸朗: 通信網オペレーションシステムにおけるネットワーク構成管理データベース, 信学会秋季全国大会, B-254 (1989).

[63] Yokoo, M., Durfee, E. H., Ishida, T., and Kuwabara, K.: Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving, *Proc. 12th International Conference on Distributed Computing Systems*, pp. 614–621 (1992).

[64] 横尾真, エドモンド H. ダーフィ, 石田亨, 桑原和宏: 分散制約充足による分散協調問題解決の定式化とその解法, 電子情報通信学会論文誌 D-I, Vol. 75, No. 8, pp. 704–713 (1992).

[65] Zlotkin, G. and Rosenschein, J. S.: Cooperation and Conflict Resolution via Negotiation Among Autonomous Agents in Noncooperative Domains, *IEEE Trans. Systems, Man and Cybernetics*, Vol. 21, No. 6, pp. 1317–1324 (1991).

# 原著論文

## 学術論文

(1) 桑原和宏, Lesser, V. R.: マルチステージネゴシエーションにおけるゴール間競合の検出, 情報処理学会論文誌, Vol. 32, No. 10, pp. 1269–1280 (1991).

(2) Conry, S. E., Kuwabara, K., Lesser, V. R., and Meyer, R. A.: Multistage Negotiation for Distributed Constraint Satisfaction, *IEEE Trans. SMC*, Vol. 21, No. 6, pp. 1462–1477 (1991).

(3) 桑原和宏: マルチステージネゴシエーションにおける探索戦略の評価, 情報処理学会論文誌, Vol. 34, No. 7, pp. 1638–1649 (1993).

(4) 桑原和宏, 石田亨, 大里延康: AgenTalk: マルチエージェントシステムにおける協調プロトコル記述, 電子情報通信学会論文誌 B-I, Vol. 79, No. 5, pp. 346–354 (1996).

## 国際会議，ワークショップ

(5) Kuwabara, K. and Lesser, V. R.: Extended Protocol for Multistage Negotiation, *Proc. 9th Workshop on Distributed Artificial Intelligence*, pp. 129–161 (1989).

(6) Kuwabara, K., Ishida, T., and Nishibe, Y.: Market-Based Distributed Resource Allocation: Equilibratory Approach, *IJCAI-93 Workshop on Artificial Economics* (1993).

(7) Kuwabara, K. and Ishida, T.: Equilibratory Approach to Distributed Resource Allocation: Toward Coordinated Balancing, in Castelfranchi, C. and Werner, E. eds., *Artificial Social Systems, MAAMAW'92*, Lecture Notes in AI 830, pp. 133–146, Springer-Verlag (1994).

(8) Kuwabara, K., Ishida, T., and Osato, N.: AgenTalk: Coordination Protocol Description for Multiagent Systems, *Proc. First International Conference on Multi-Agent Systems (ICMAS '95)*, p. 455 (1995).

(9) Kuwabara, K., Ishida, T., and Osato, N.: AgenTalk: Describing Multiagent Coordination Protocols with Inheritance, *Proc. 7th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '95)*, pp. 460–465, (1995).

(10) Kuwabara, K.: Meta-Level Control of Coordination Protocols, *Proc. Second International Conference on Multi-Agent Systems (ICMAS '96)*, pp. 165–172, (1996).

## 研究会など

(11) 桑原和宏: マルチステージネゴシエーションによる分散資源割当, 1990 年代の分散処理シンポジウム, pp. 87–96, 情報処理学会 (1990).

(12) 桑原和宏, 石田亨: 分散資源割当における共生的アプローチ: 通信網の資源管理を目指して, 信学技報 AI 92-67 (1992).

(13) 桑原和宏, 石田亨, 大里延康: AgenTalk: マルチエージェントシステムにおける協調プロトコル記述, 信学技報 AI 94-56 (1995).

(14) 桑原和宏, 篠原拓嗣, 大里延康, 石田亨: 協調プロトコル記述言語 AgenTalk の実現, 信学技報 AI 95-18 (1995).

(15) 桑原和宏, 上野磯生, 吉田仙: マルチエージェントにおける協調プロトコル記述とその応用, 第 5 回通信ソフトウェア研究会, pp. 14–21, 電子情報通信学会通信ソフトウェア時限研究専門委員会 (1996).



## 関連共著論文 (学術論文, 国際会議, ワークショップ)

(16) Nishibe, Y., Kuwabara, K., and Ishida, T.: Effects of Heuristics in Distributed Constraint Satisfaction: Towards Satisficing Algorithms, *Proc. 11th International Workshop on Distributed Artificial Intelligence*, pp. 285–302 (1992).

(17) 西部喜康, 桑原和宏, 石田亨, 横尾真: 分散制約充足の高速化と通信網回線設定への適用, 電子情報通信学会論文誌 D-II, Vol. 76, No. 10, pp. 2204–2214 (1993).

(18) Nishibe, Y., Kuwabara, K., Suda, T., and Ishida, T.: Distributed Channel Allocation in ATM Networks, *Proc. GLOBECOM '93*, pp. 417 – 423 (1993).

(19) Yokoo, M., Durfee, E. H., Ishida, T., and Kuwabara, K.: Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving, *Proc. 12th International Conference on Distributed Computing Systems*, pp. 614–621 (1992).

(20) 横尾真, エドモンド H. ダーフィ, 石田亨, 桑原和宏: 分散制約充足による分散協調問題解決の定式化とその解法, 電子情報通信学会論文誌 D-I, Vol. 75, No. 8, pp. 704–713 (1992).

(21) Nishibe, Y., Kuwabara, K., Yokoo, M., and Ishida, T.: Speed-Up and Application Distributed Constraint Satisfaction to Communication Network Path Assignments, *Systems and Computers in Japan*, Vol. 25, No. 12, pp. 54–67 (1994).

## その他 (解説, 単行本など)

(22) 桑原和宏, 石田亨: 分散人工知能 (2): 交渉と均衡化, 人工知能学会誌, Vol. 8, No. 1, pp. 17–25 (1993).

(23) 桑原和宏: 交渉とその周辺, 情処研報 AI 92-3, pp. 21–30 (1994).

(24) 桑原和宏: ネットワークとマルチエージェント, 第 9 回人工知能学会全国大会チュートリアル講演, 人工知能学会 (1995).

(25) 桑原和宏: マルチエージェントのネットワークへの応用, 信学技報 IN 95-75 (1995).

(26) Kuwabara, K., Ishida, T., Nishibe, Y., and Suda, T.: An Equilibratory Market-Based Approach for Distributed Resource Allocation and Its Applications to Communication Network Control, in Clearwater, S. H. ed., *Market-Based Control: A Paradigm for Distributed Resource Allocation*, World Scientific Publishing (1996).

(27) 桑原和宏: 市場モデル, 石田亨, 片桐 恭弘, 桑原 和宏共著, "分散人工知能", 第 8 章, コロナ社 (1996).

(28) 石田亨, 桑原和宏: 分散人工知能 (1): 協調問題解決, 人工知能学会誌, Vol. 7, No. 6, pp. 945–954 (1992).

(29) 石田亨, 桑原和宏: 分散人工知能, 情報処理学会 (編), 情報処理ハンドブック, 13 編 5 章 1, pp. 1466–1468, オーム社 (1995).

# 付録 A

# AgenTalk 仕様概要

## A.1 はじめに

第 6 章で述べた協調プロトコル記述言語 AgenTalk は，メタレベル制御機能を除いた基本的な機能を実装した初期バージョンをフリーソフトとして公開している[1]．この付録では，この初期バージョンについてその言語仕様の概略を述べる．詳しくはリファレンスマニュアル [27] を参照されたい．

現在，AgenTalk は Common Lisp 上で実装されており[2]，基本的には Common Lisp の機能をそのまま使うことができる．ここで述べる言語機能は Common Lisp のマクロないしは関数として定義される．

個々のエージェントは Lisp 処理系のマルチプロセスの機能を用いて Lisp における一つのプロセス (スレッド) として実装されている (図 A.1)．同じ Lisp システム内にあるエージェント同士は (共有) メモリを介した通信を行い，また，別の Lisp システムにあるエージェントとは TCP/IP を用いた通信を行う．さらに，C 言語からメッセージの読み書きができるインタフェースも合わせて作成し，C で書かれたプログラムとの通信も実現している (図 A.2)．このエージェント間の通信機能を用いて，テレオーガニゼーション [19] の実現を目指したエージェントネットワーク Socia が構築されており，卓上電子会議支援システムの実現に用いられている [22].

Common Lisp の実装では次のようなパッケージ (package) を定義している．

[1] `http://www.cslab.tas.ntt.jp/at/`

[2] ただし Allegro Common Lisp (ACL) version 4.2 および Macintosh Common Lisp (MCL) 3.0 以降のマルチプロセス (スレッド) の拡張機能を用いている．

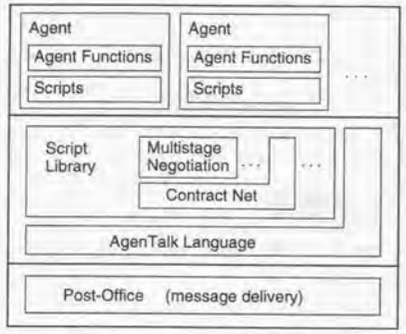



図 A.1: AgenTalk の構成

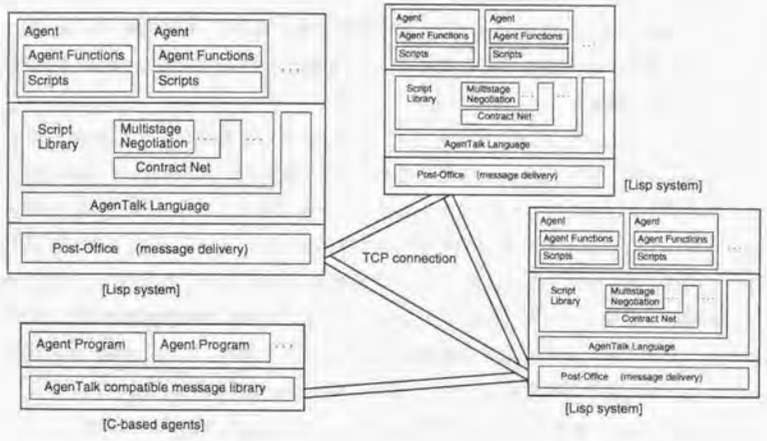



図 A.2: ネットワークを用いた構成例



- agentalk (nickname at): AgenTalk で定義する関数，マクロ，変数はこのパッケージのシンボルとして定義される．
- agentalk-user (nickname atu): AgenTalk のアプリケーションを記述するデフォールトのパッケージ．agentalk パッケージを use している．
- agentalk-message (nickname atm): メッセージのクラス名，スロット名はこのパッケージのシンボルとして定義される．パッケージを統一することで，他の Lisp システムとの通信においてメッセージの読み込み時のパッケージの差異を吸収するためである．
- agentalk-internal (nickname ati): AgenTalk の内部関数はこのパッケージのシンボルとして定義される．このパッケージを独自に設けた理由は AgenTalk の内部構成を複数のパッケージで構成した時にそれら間で共通に使われるシンボルをこのパッケージ内に定義できるようにするためである．

## A.2 エージェント

### A.2.1 エージェントクラス

エージェントは CLOS (Common Lisp Object System) のインスタンスとして定義される．エージェントのクラスとして agent と script-agent が定義されている．アプリケーションプログラムはこれらのクラスを使うか，また，これらのクラスをスーパクラスに持つクラスを定義することになる．

agent [*Class*]

agent がエージェントのベースとなるクラスであり，Lisp システム内のプロセス (スレッド) 生成，メッセージのやり取りなどエージェントの基本的機能を定義している．

script-agent [*Class*]

クラス script-agent は後述するスクリプトの機能を提供する．スクリプト機能を使

うエージェントはこのクラス，またはそのサブクラスのインスタンスでなければならない．なお，クラス script-agent は agent のサブクラスになっている．

### A.2.2 エージェント名

エージェントはユニークな名前を持つ．メッセージの相手先はエージェントの名前によって指定する．この時，エージェントがどこにあるかは意識しなくてもよい．

エージェントの名前はエージェントが所属するロケーション (location) の名前とそのロケーションの中でのローカルな名前の組み合わせとして，*local-name*@*location-name*の形式で表現される．各ロケーションごとにはメッセージの配送を行う post-office という機構が存在する．すなわち，各 post-office ごとに *location-name*が与えられる．post-office にエージェントを登録することによって他のエージェントとメッセージのやり取りを行うことができる．

Common Lisp 上の実装では一つの Lisp システムごとに post-office が存在し，名前 (*location-name*) を与える．同一のロケーション内 (すなわち同一の Lisp システム内) においては *location-name*を省略することができる．

なお，エージェント名 (*local-name*, *location-name*共に) は大文字小文字の区別はしない．

### A.2.3 エージェント関連の関数

エージェントは CLOS におけるインスタンス生成の総称関数 (generic function) である make-instance を用いて生成する．エージェント名はこの関数の呼び出し時に:name のキーワードの引数で与える．エージェントの名前は文字列 (string) かシンボル (symbol) で，シンボルが渡された時はそのシンボル名 (symbol-name) が用いられる．

add-agent *agent* &optional *process-run-function* [*Generic Function*]
&rest *args*

add-agent (*agent* agent) &optional *process-run-function* [*Primary Method*]
&rest *args*



*agent*を post-office に登録する．これによってメッセージのやり取りが可能になる．さらに，このエージェントに対応して新たな Lisp 内のプロセス (スレッド) が作られる．この時，*process-run-function* (デフォールト値は agent-toplevel) が *args*で与えられた引数とともに起動される．

agent-toplevel *agent* &rest *args* [*Generic Function*]

agent-toplevel (*agent* agent) &rest *ignored* [*Primary Method*]

agent-toplevel (*agent* script-agent) *script-name* [*Primary Method*]
&rest *args*

agent-toplevel はエージェントのプロセス (スレッド) のトップレベルのループを提供している．クラス agent に関しては，エージェントのメッセージキューからメッセージをとりだし，対応するメッセージハンドラを起動することを繰り返す．

script-agent に対するメソッドでは *script-name*を引数として与えることにより，エージェント起動時にスクリプト *script-name*を起動することができる．*args*は起動するスクリプトに対する引数となる (run-script 参照)．

find-agent *name* &optional *errorp* [*Function*]

この関数は *name*の名前をもつエージェント (CLOS のインスタンス) を返す．*name*は文字列またはシンボルである．名前 *name*のエージェントが存在しない場合，*errorp*が nil でない時はエラーとなり，*errorp*が nil(デフォールト) の時は nil が返る．

agent-name *agent* [*Function*]

この関数は *agent*の名前を文字列として返す．

## A.3 メッセージ

エージェント間で交換されるメッセージはクラス message またはそのサブクラスのインスタンスとして実装される．

message [*Class*]

message はメッセージのクラスのルートとなるクラスである．

define-message-class *class* ( [*super*] ) (*slot-decl**) &rest *options* [*Macro*]

*slot-decl* ::= (*slot-name* [:type *type*] ) | *slot-name*

*options* ::= (:documentation *doc-string*)

メッセージクラス *class* を定義する．そのスーパクラスを *super* で指定する．*super* が指定されない時は message が使われる．*type* がスロット値の型を指定するが，現バージョンではこの情報は使われていない．

例:

```
(define-message-class announcement ()
  ((specification :type string)
   (expiration :type integer)))
```

ここで，メッセージクラス名，スロット名は agentalk-message パッケージのシンボルとなる．

create-message *class receiver sender* &rest *slot-values* [*Function*]

*slot-values* ::= {*slot-name value*}*

メッセージクラス *class* のインスタンスとなるメッセージを生成して返す．*receiver* はメッセージの宛先となるエージェントの名前，またはブロードキャストの場合は * を指定する．*sender* はメッセージの送り手を指定する．*slot-name* はキーワード (: で始まるシンボル) で指定する．

例:

```
(create-message 'announcement "*" "ME"
                :specification "TASK1" :expiration 3003044053)
```



send-message *message* &rest *args* [*Generic Function*]

send-message (*message* message) &rest *ignored* [*Primary Method*]

send-message (*class* symbol) *receiver sender* [*Primary Method*]
&rest *slot-values*

message クラス (またはそのサブグラス) のインスタンス *message* が引数として渡された時は *message* が実際に送信される．第一引数としてクラス名 (シンボル) が渡された時は残りの引数が create-message の関数に渡され，メッセージクラス *class* のインスタンスが生成され，生成されたメッセージが実際に送信される．send-message は送信したメッセージを返す．

get-message-sender *message* [*Function*]

*message* の送信元を返す．

get-message-slot-value *message slot-name* [*Function*]

この関数は *message* のスロット *slot-name* の値を返す．もし，その名前のスロットが存在しない時はエラーとなる．

注: メッセージは CLOS のインスタンスとして実装されているので，slot-value の総称関数を用いてスロットにアクセスすることができる．しかし，スロット名は agentalk-message のパッケージのシンボルなので，get-message-slot-value を使う方が便利である．

## A.4 メッセージパタン

エージェントは基本的には送られてきたメッセージに対して何らかの行動をとることを繰り返す．したがって，どのようなメッセージが来たらどのような行動をとるかを記述する必要がある．AgenTalk ではメッセージパタンを用いてどのようなメッセージかを指定することができる．また，メッセージパタンにメッセージハンドラを対応づけることができ，メッセージハンドラとしてメッセージパタンにマッチするメッセージが到着した時にエージェントがとるべき行動を記述することができる．

メッセージが到着した時は，アクティブなメッセージパタンの中からマッチするメッセージパタンを見つけ，マッチしたメッセージパタンに対応づけられたメッセージハンドラを起動することになる．

`define-message-pattern` *name-and-options pattern handler-body* [*Macro*]

*name-and-options* ::= *name* | (*name options*)
*options* ::= (`:documentation` *doc-string*)

*name*の名前を持つメッセージパタン (*pattern*) とそれに対応したメッセージハンドラ (*handler-body*) を定義する．*pattern*のシンタックスは以下の通りである．

*pattern* ::= *class receiver sender* [*priority*] *slots*
*class* ::= *symbol*
*receiver* ::= *wildcard-name* | *name* | ({*name*}$^+$) | *pattern-var*
*sender* ::= *wildcard-name* | *name* | ({*name*}$^+$) | *pattern-var*
*wildcard-name* ::= "*" | *
*name* ::= *symbol* | *string*
*priority* ::= *number* | *lisp-form-number*
*slots* ::= {*slot-name value-pattern*}*
*slot-name* ::= *keyword*
*value-pattern* ::= *pattern-var* | *pattern-form* | *lisp-form-quoted*
| (*value-pattern* {*value-pattern*}*)
*pattern-var* ::= (`:?` [*pattern-var-name*]
[`:value-restriction` *value-restrictions*]
[`:form-restriction` *lisp-form*]
*simple-pattern-var* ::= (`:?` *pattern-var-name*)
*pattern-var-name* ::= *symbol*
*value-restrictions* ::= ({*value-restriction*}$^+$)
*value-restriction* ::= (`AND` {*value-restriction*}$^+$) | (`OR` {*value-restriction*}$^+$)
| (`NOT` *value-restriction*) | *value-rest-prim*
*value-rest-prim* ::= ({`>` | `>=` | `=` | `<` | `<=`} *lisp-form-number*) | *lisp-form*
*pattern-form* ::= (`:!` *lisp-form*)

ここで *lisp-form*は副作用のない Lisp のフォームを表す．ただし，*lisp-form*の中に値が束縛されている *simple-pattern-var*を使うことができる．なお，*lisp-form*はパタンマッチの時に評価される．*lisp-form-number*は評価結果が数になる *lisp-form*を表す．*lisp-form-quoted*はパタンマッチ時には評価されない．優先度 (priority) はシステム内部では整数値として扱われる．整数値以外の値が指定された時は `truncate` 関数を用いて，整数値に切り捨てられる．メッセージのスロット値の比較には Common Lisp の `equal` 関数が用いられる．

*handler-body*は基本的には *lisp-form*である．ただし，その中では `msg`，`self`，`context` のシンボルが変数として暗黙のうちに宣言されている．`msg` には到着したメッセージが束縛され，`self` にはメッセージを受けとったエージェントが束縛される．`context` はスクリプトの実行コンテキストが束縛される (実行コンテキストに関してはスクリプトの節を参照)．さらに *handler-body*にはメッセージパタン中の *pattern-var*が現れることができる．

例:

```
(define-message-pattern pattern1
    (announcement * coordinator 1
                  :name "assessment"
                  :time (:? x :value-restriction ((> 10))))
  (format t "Announcement ~s with time ~s is received by ~s"
          msg (:? x) self))
```

この例はメッセージクラス `announcement` のメッセージが `coordinator` という名前のエージェントから届いた時，`name` スロットの値が `"assessment"`であり，かつ `time` スロットの値が 10 より大きい時にマッチするメッセージパタンとそれに対応するメッセージハンドラ (`(format ...)`) を定義している．

`wait-for-message` *agent message-pattern* [*Generic Function*]
`&optional` *timeout*
`&key :when-message-arrived :when-timeout`
`wait-for-message` (*agent* `agent`) [*Primary Method*]
(*message-pattern* `message-pattern`) `&optional` *timeout*
`&key :when-message-arrived :when-timeout`

`wait-for-message` は *message-pattern*のメッセージパタンにマッチするメッセージ

が到着するまでエージェントのプログラムの実行を中断する．*timeout*が指定された時は，*timeout*秒過ぎた時にはメッセージの到着がなくてもエージェントの実行は再開する．*timeout*として，nilが渡された時はタイムアウトしない．メッセージパタンにマッチするメッセージが到着した時は，到着したメッセージと*agent*を引数として:when-message-arrivedで指定された関数が起動される．また，タイムアウトが起きた時，*agent*を引数として:when-timeoutで指定された関数が起動される．

define-message-pattern-template *name-and-options lambda-list* [*Macro*]
*pattern*

*name-and-options* ::= *name* | (*name options*)
*options* ::= (:documentation *doc-string*)

このマクロは*name*という名前のついたメッセージパタンテンプレートを定義する．メッセージパタンテンプレートは一部しか違わないメッセージパタンを一つのテンプレートから作るために用意されている．

メッセージパタン中の値が違うメッセージパタンを複数作成する時には，値を変更する部分をメッセージパタンテンプレートの定義において*lambda-list*中の変数として定義し，その変数を*lisp-form*の中で使用すればよい．メッセージパタンテンプレートから値が異なるメッセージパタンを作成する時はメッセージパタン作成時に違う値を与えることになる．

なお，テンプレートから作成されたメッセージパタンについては，パターンマッチにおける*pattern*内の*lisp-form*の評価が，*lambda-list*中の変数がメッセージパタン作成の関数を呼び出した時の引数の値に束縛された環境で行われることになる．

以下にwait-for-message関数と共に用いる例を示す．このメッセージパタンテンプレートからはreplyクラスのメッセージにマッチするメッセージパタンを作成できる．この時，receiver-name, sender-name, :reply-toスロットの値message-idならびに優先度をメッセージパタンテンプレートからメッセージパタンを作成する時に引数として指定することができる．



例:

```
(define-message-pattern-template wait-for-reply
  (receiver-name sender-name message-id
    &optional (priority *wait-for-reply-priority*))
  (reply (:! receiver-name) (:! sender-name) (:! priority)
         :reply-to (:! message-id)))
```

create-message-pattern-from-template *template-name* [*Function*]
&rest *args*

*template-name*の名前のメッセージパタンテンプレートからメッセージパタンを作る．*args*で渡された引数によってメッセージパタンが初期化される．

set-message-pattern-priority *message-pattern new-priority* [*Function*]

*message-pattern*の優先度を*new-priority*にセットする．

remove-message-pattern *message-pattern* [*Generic Function*]
remove-message-pattern (*message-pattern* message-pattern) [*Primary Method*]
remove-message-pattern (*message-pattern-name* symbol) [*Primary Method*]

このメソッドは*message-pattern*または*message-pattern-name*の名前を持つメッセージパタンを削除する．

initialize-message-patterns &optional *clear-all-p* [*Function*]

メッセージパタンに関する機能を初期化する．*clear-all-p*がnilでない時はシステム定義のものを含めすべてのメッセージパタンが削除される．AgenTalk システム自体をデバッグするのでなければ，*clear-all-p*はnilとする．

## A.5 スクリプト

スクリプト (script) によってエージェント間の協調プロトコルが表現される．スクリプトは拡張有限状態機械をベースにしている．拡張有限状態機械は define-script で宣言され，拡張有限状態機械 (スクリプト) の各状態は define-script-state で定義される．状態遷移規則はそれが有効となる状態に付随して define-script-state の中で定義される．さらにスクリプトに付随して名前付き状態遷移規則を定義できる．状態定義の中でこの状態遷移規則の名前を指定することで，その状態の中で名前付き状態遷移規則を有効にすることができる．なお，スクリプト機能を使うためにはエージェントのクラスは script-agent またはそのサブクラスである必要がある．

スクリプトの機能はアクティブなメッセージパタンを動的に切り替えることで実現している．すなわち，スクリプトにおける状態はアクティブなメッセージパタンの集合として実装され，状態間の遷移は有効となるメッセージパタンの集合を切り替えることで実現されている．したがって，状態遷移を記述したスクリプトの定義は基本的にはメッセージパタンと対応するメッセージハンドラの定義に分解されることになる．

### A.5.1 スクリプト定義

`define-script` *script-name lambda-list* {*script-options*}* [*Macro*]

*script-name* ::= *symbol*

*script-options* ::= `:initial-state` *initial-state-name*
| `:script-vars` ({*script-var* | (*script-var initial-value*)}*)
| `:inherits-from` *script-name*
| `:initial-active-states` *state-name-list*
| `:on-entry` *rule-lisp-form*
| `:documentation` *doc-string*

スクリプト *script-name*を定義する．*lambda-list*はスクリプト起動の関数 (`run-script`) の引数となる．*lambda-list*は Common Lisp と同じシンタックスをとり，`&optional`, `&key`, `&rest` を用いることができる．*lambda-list*の中に現れる変数はスクリプト変数として取り扱われる．

`:script-vars` を用いてスクリプト変数を定義できる．*initial-value*が指定された時はスクリプト起動時に *initial-value*が評価され，スクリプト変数はその評価結果に束縛される．*initial-value*において *lambda-list*の中に現れるスクリプト変数にスクリプト変数アクセスマクロ (`$`) を用いてアクセスできる．*initial-value*が指定されていない時は `nil` が初期値となる．

`:inherits-from` はスクリプトの継承元のスクリプト (スーパスクリプト) を指定する．継承元のスクリプトより，状態，スクリプト変数，スクリプト関数の定義を継承する．

`:on-entry` が指定されている時は初期状態に移行する前に`:on-entry` で定義されている *rule-lisp-form*が実行される．この中で `call-super` 関数を用いてスーパスクリプトの `on-entry` の定義を呼び出すこともできる．

`:initial-state` で初期状態を指定する．なお *initial-state-name*は評価される．

また，現バージョンでは実験的な機能として複数の状態を同時にアクティブにすることができる．アクティブな状態に含まれる状態遷移規則がスクリプト実行においてはパタンマッチの対象となる．`:initial-active-states` は最初にアクティブにすべき状態を指定する．

例:

```
(define-script cnet-manager (task)
  :initial-state 'start
  :script-vars (bid-queue contract-id))
```

なお，`nil` はスクリプト名には用いられない．

`define-script-rule` (*rule-name script-name*) [*Macro*]

`&key` *script-vars-used rule documentation*

名前付き状態遷移規則を定義する．状態遷移規則の定義中で`:script-vars-used` オプションで指定されたスクリプト変数はスクリプト変数アクセスマクロ (`$`) を用いずにアクセスすることができる．

*rule* ::= (:when *condition* :do *action*)
*condition* ::= [*msg-condition* | *timeout-condition*] *script-var-condition*
| *msg-condition* | *timeout-condition*
*msg-condition* ::= (msg *rule-message-pattern*)
*timeout-condition* ::= (timeout *timeout*)
*script-var-condition* ::= (test *rule-lisp-form*)
*rule-message-pattern* ::= *message-class* [*sender*] {*slot-name rule-pattern-value*}*
*slot-name* ::= *keyword*
*timeout* ::= *rule-lisp-form*
*action* ::= {*rule-action-lisp-form*}*

*rule-message-pattern*のシンタックスは define-message-pattern 中の *message-pattern* と以下の点を除いて同一である．

- *receiver*はスクリプトを起動したエージェント名が指定されたものと扱われ，*receiver*は省略される．
- *sender*は省略することができる．この時は "*"が指定されたものと扱われる．
- *priority*を指定することはできない．優先度はスクリプト実行制御用に内部で用いられている．

*timeout*は数が指定されなければいけない．タイムアウトの条件は状態遷移した後 *timeout*秒後に成立する．*rule-lisp-form*は *lisp form*と同じである．ただし，スクリプト変数アクセスマクロ ($) とスクリプト関数 / エージェント関数呼出しマクロ (!) を用いることができる．また，*rule-lisp-action-form*では $ と ! に加えて，goto-state, invoke-script, exit-script が使用可能である．

define-script-state (*state-name script-name*) {*script-state-options*}* [*Macro*]



*script-state-options* ::= :script-vars-used ({*script-var*}*)
| :on-entry *rule-lisp-form*
| :rules ({*rule-spec*}*)
| :priority *number*
| :documentation *doc-string*

スクリプト *script-name*の状態を定義する．:script-vars-used の中で指定されたスクリプト変数は:on-entry の *rule-lisp-form*中，および *rule-spec*中でスクリプト変数アクセスマクロ ($) を用いることなく通常の変数と同様に扱うことができる．:on-entry が定義された時は，この状態に遷移した時に:on-entry で定義された *rule-lisp-form*が起動される．:rules オプションにより，この状態で有効となる状態遷移規則を定義する．さらに:name オプションで名前を指定することで名前付き状態遷移規則 (define-script-rule で定義されるもの) をこの状態で有効にすることができる．すなわち，

*rule-spec* ::= *rule* | (:named *rule-name* [:here] )

となる．ここで，:here オプションは名前付き状態遷移規則の継承がどのように扱われるかを指定する．仮にスクリプト SuperScript を継承してスクリプト SubScript が定義されたとする．さらに両方のスクリプト共に Rule1 という名前付き状態遷移規則が定義されているとする．さらにスクリプト SuperScript が状態 State1 を定義し，スクリプト SubScript に継承されているとする．ここでスクリプト SubScript において，状態 State1 の定義 (これはもともとは SuperScript で定義されている) 中における (:name Rule1) は SubScript で定義されている名前付き状態遷移規則を指す．しかし，(:name Rule1 :here) の場合は SuperScript で定義された名前付き状態遷移規則を指す．

:priority は状態の優先度を表す．これは複数の状態を同時にアクティブにした時，どの状態遷移規則を優先するかの状態間の優先関係を示す．また，nil は状態名として使うことができない．

例:

```
(define-script-state (start cnet-manager)
  :script-vars-used (task contract-id)
  :on-entry (progn (setf contract-id (! announce-task))
                   (goto-state 'announced)))

(define-script-state (announced cnet-manager)
  :script-vars-used (task bid-queue contract-id)
  :rules
  ((:when (msg bid :contract-id !contract-id)
    :do (if (! send-award-immediately-if-possible)
            (goto-state 'success)
            (push msg bid-queue)))
   (:when (timeout (task-expiration task))
    :do (if (! send-award-if-possible)
            (goto-state 'success)
            (goto-state 'failure)))))
```

`remove-script-state` *script-name state-name* [*Function*]

この関数はスクリプト *script-name*から状態 *state-name*を取り除く．この状態とともに定義されている状態遷移規則 (名前付き状態遷移規則を除く) も同時に取り除かれる．

`run-script` *agent script-name* `&rest` *args* [*Generic Function*]

`run-script` (*agent* `script-agent`) *script-name* `&rest` *args* [*Primary Method*]

スクリプトを起動する．ここで引数は `define-script` で定義した *lambda-list*と合致する必要がある．この関数はスクリプトの実行コンテキストを返す．

注: この関数はトップレベル (親のコンテキストが存在しない時) のスクリプトを起動する時にのみ使う．状態遷移規則の中から新たにスクリプトを起動する時はルールマクロ `invoke-script` を用いる．



## A.6 スクリプト実行コンテキスト

スクリプトが起動された時にはスクリプト実行コンテキストが生成される．コンテキストはスクリプト実行に関する種々の情報が保持されている．次のマクロを用いてコンテキストで保持されている情報にアクセスすることができる．以下に示すマクロの引数 *context*のデフォールト値は現在のコンテキストである．

`current-state` `&optional` *context* [*Macro*]

`current-state` は現在の状態の名前を返す．すでにスクリプトの実行が終了している時は `nil` を返す．

`exited-state` `&optional` *context* [*Macro*]

`exited-state` はスクリプトの実行が終了した時の最後の状態名を返す．まだ，スクリプトが実行中の時は `nil` が返る．

`parent-context` `&optional` *context* [*Macro*]

`parent-context` は親のコンテキストを返す．

`child-contexts` `&optional` *context* [*Macro*]

`child-contexts` は子のコンテキストのリストを返す．

### A.6.1 状態遷移規則

状態遷移規則の中で次のマクロ (ルールマクロ) を用いることができる．

`$` *script-var-name* `&optional` *context* [*Rule Macro*]

スクリプト変数をアクセスするマクロである．スクリプト変数に値を設定するためには setf を用いる．

! *function-name* &rest *args* [*Rule Macro*]

エージェント関数 *function-name*を引数 *args*とともに起動する．エージェント関数が存在しない時は同名のスクリプト関数が起動される．もし，エージェント関数もスクリプト関数も存在しない時はエラーとなる．

次のマクロを状態遷移規則の実行部 (*rule-action-lisp-form*) で用いることができる．

goto-state *state-name* &optional *context* [*Rule Action*]

状態 *state-name*へ状態遷移する．

invoke-script *script-name* &rest *args* [*Rule Action*]

新たにスクリプトを起動する．スクリプトを起動した時点で invoke-script 自体の実行は終る．新たに生成されたコンテキストを返す．

call-script *script-name* &rest *args* [*Rule Action*]

新たにスクリプトを起動する． invoke-script と違い， call-script でスクリプトを起動した時は起動されたスクリプトが exit-script により実行を終了した時点で初めて call-script 自体の実行が終了する． call-script の返却値は exit-script において *result*として渡された値である．

exit-script &optional *result context* [*Rule Action*]

*context*で指定したコンテキストのスクリプトの実行を終了する． *context*のデフォルト値は現在のコンテキストである．もし，コンテキストの子のコンテキストが無ければ自分自身のコンテキストも親のコンテキストから削除される． *result*のデフォルト値は nil である．



### A.6.2 エージェント / スクリプト関数

スクリプトを部分的にカスタマイズするためにエージェント関数のインタフェースが設けられている．スクリプト定義の中からエージェント / スクリプト関数が ! で起動される時，まず，エージェント関数が探される．もし，見つかった場合はその関数が呼ばれ，そうでない時は同名のスクリプト関数が探される．(このようにスクリプト関数はエージェント関数に対するデフォールトの振舞いを定義している．したがって，スクリプトのカスタマイズは，エージェント関数を定義するか，またはスクリプト関数の定義だけを変えた新たなスクリプトを元のスクリプトを継承しつつ定義することによって行なえる．)

define-agent-function (*function-name agent-name*) *lambda-list* {*rule-form*}* [*Macro*]

エージェント関数を定義する[3]．

define-script-function (*function-name script-name*) *lambda-list* {*rule-form*}* [*Macro*]

スクリプト *script-name*におけるスクリプト関数を定義する．

with-script-vars ({*script-var*}*) {*rule-form*}+ [*Macro*]

*rule-form*の中では *script-var*で指定されたスクリプト変数はスクリプト変数アクセスマクロ ($) を用いずにアクセスできる． このマクロはスクリプト定義，スクリプト状態定義ならびに状態遷移規則の定義の中で用いることができる．

[3]現バージョンではエージェント関数の名前の空間はエージェントごとに一つしかない．したがって，異なるスクリプトから同名のエージェント関数が呼ばれる場合でも，一つのエージェント関数しか定義できないことになる．この点については将来のバージョンで改善する予定である．

`call-super &rest` *args* [*Function*]

スクリプト関数定義の中で，継承元のスクリプトの同名のスクリプト関数を呼び出すのに用いる．もし，*args*が与えられない時はもとのスクリプト関数の呼び出しにおける引数が渡される．これは CLOS における `call-next-method` と同様である．

`remove-agent-function` *agent-or-name function-name* [*Generic Function*]
`remove-agent-function` (*agent* `script-agent`) *function-name* [*Primary Method*]
`remove-agent-function` (*agent-name* `symbol`) *function-name* [*Primary Method*]

エージェント関数の定義を削除する．

`remove-script-function` *script-name function-name* [*Function*]

スクリプト関数の定義を削除する．

## A.7 リードマクロ

メッセージパタン中の *pattern-var*と *pattern-form*の記述を容易にするための Common Lisp のリードマクロ (read macro) 機能を利用した簡略記法を用意している．

この簡略記法では? の文字が *pattern-var*の始まりを表している．また，！の文字が *pattern-form*の始まりを表している．*pattern-var*の中では & の文字が *value-restriction*の始まりを表し，また，！の文字が *form-restriction*の始まりを表している．具体的なシンタックスは以下の通りである．

*pattern-var-read* ::= ?*pattern-var-name*{&*value-restriction*}* [!*form-restriction*]

*pattern-form-read* ::= !*lisp-form*

ここで *pattern-var-read*中の & と！の文字の前には空白が入ってはならない．

*pattern-var-read*は読み込み時に *pattern-var*に展開され，また，*pattern-form-read*も読み込み時に *pattern-form*に展開される．



例:

```
?var&(> 3)&(< 5)!(test ?var)
   ⇒ (:? var :value-restriction ((> 3)(< 5))
             :form-restriction (test (:? var)))
!(test-form ?var)
   ⇒ (:! (test-form (:? var)))
```

`use-readtable &optional` *readtable-name* [*Macro*]

上記のリードマクロを使うためには Common Lisp の `*readtable*` を設定する必要がある．これを行なうためには `use-readtable` のマクロを使用する． 一般にはプログラムファイルの中で (`in-package ...`) の直後にこのマクロを呼べばよい．*readtable-name*は名前つきの readtable を機能を持っている Lisp 処理系 (例えば Allegro Common Lisp Ver. 4.2) で用いられる．現バージョンでは:`agentalk` の名前で定義されている．

## A.8 デバック機能

現バージョンの AgenTalk はスクリプト実行のトレース，ブレークポイントの設定などの簡単なデバッグ機能を提供している．さらに X window system においてエージェントごとに異なる window に出力を出す `interactor` という外部プログラムを利用することができる．

### A.8.1 トレース

`trace-script` {*script-spec*}* [*macro*]

*script-spec* ::= *script-name* | (*script-name* {*trace-option*}*)
*trace-option* ::= `:break` *boolean* | `:trace` *boolean*

スクリプト実行のトレース / ブレークポイントの設定を行なう．また，`:break` オプションを指定することにより，スクリプトが起動された時に Common Lisp のデバッガを起動することができる．`:trace` オプションを指定することにより，スクリプト

が起動された時にメッセージが出力される．何のオプションも指定されない時 (すなわち *script-name*のみ指定された時) は:trace t が仮定される．

*script-spec*が省略された時は現在トレースの対象になっているスクリプトのリストが返る．

untrace-script {*script-name*}* [*Macro*]

スクリプトのトレースの設定を解除する．*script-name*が省略された時はすべてのスクリプトのトレースの設定が解除される．

trace-state-transition [*script-name* {*state-spec*}*] [*Macro*]
*state-spec* ::= *state-name-or-all* | (*state-name-or-all* {*trace-option*}*)
*state-name-or-all* ::= *state-name* | :all
*trace-option* ::= :break *boolean* | :trace *boolean*

状態遷移のトレース / ブレークポイントの設定を行なう．:break オプションが設定された時，スクリプト *script-name*で状態 *state-name*への状態遷移が起きた時に Common Lisp のデバッガを起動する．:trace オプションが指定された時は代わりにメッセージが出力される．*state-name*の代わりに:all が指定された時はすべての状態遷移がトレース / ブレークの対象となる．もし，引数が渡されない時はは現在トレースの対象となっている状態のリストが返る．また，*script-name*だけが指定された時はスクリプト *script-name*に関してトレースの対象となっている状態のリストが返る．

untrace-state-transition [*script-name* {*state-name-or-all*}*] [*Macro*]

状態遷移に関するトレースの指定を取り除く．もし，引数が何も与えられない時はすべてのトレースの指定が取り除かれる．*script-name*だけが与えられた時はそのスクリプトに関連したトレースの指定のみが取り除かれる．

trace-script-function {*function-name*}* [*Macro*]



スクリプト関数 / エージェント関数のトレースを指定する．もし，*function-name*が指定されない時は現在トレースの対象になっている関数名のリストが返る．

untrace-script-function {*function-name*}* [*Macro*]

スクリプト関数 / エージェント関数のトレースの指定を取り除く．*function-name*の指定が無い時はすべてのトレースの指定が取り除かれる．

### A.8.2 Interactor

use-interactor &rest *spec* [*Function*]

*spec* ::= :interactor :host *host* :port *port* | :emacs | nil

*spec*が:interactor の時は，その後のエージェントのプロセスの *terminal-io* は interactor のプログラムに向けられることとなる．*host*と *port*は interactor のホスト名とポート番号を指定する．デフォールト値はそれぞれ *interactor-host* と *interactor-port* からとられる．

また，*spec*が:emacs の時は interactor の代わりに emacs (mule) を用いて，各エージェントの出力を異なるバッファに出力することができる．さらに，nil を指定することで，interactor の使用をやめることができる．

*interactor-host* [*Variable*]

interactor のホスト名のデフォールト値を保持する．

*interactor-port* [*Variable*]

interactor のポート番号のデフォールト値を保持する．

### A.8.3 その他の関数

`initialize &optional` *clear-system-definitions* [*Function*]

システムを初期化する．*clear-system-definitions*が `nil` でない場合はシステムで定義しているメッセージパタン，スクリプトの定義も取り除かれる．

`connect` [*Function*]

変数 `*post-office-list*` に指定された post-office との TCP コネクションを開く．

`reset &optional` *clear-message-patterns* [*Function*]

post-office との TCP コネクションを閉じ，同じ Lisp システム内のすべてのクローズしエージェントの定義を削除する．*clear-message-patterns*が `nil` でない時はすべてのメッセージパタンの定義 (システム定義のものは除く) も削除される．

`*post-office-list*` [*Variable*]

この変数に post-office に関する情報が次の形式で格納される．

`*post-office-list*` ::= ({(*post-office-location medium*)}$^+$)
*post-office-location* ::= *location-name*
*medium* ::= `:tcp-ip` *host-name port-number-or-service-name*

`*location-name*` [*Variable*]

Lisp システムのロケーション (location) の名前を保持する．