特別研究報告書

クローニングを導入した
マルチエージェントシステムの自己組織化

指導教員　　石田 亨 教授

京都大学工学部情報学科

中井 喜之

平成20年2月8日

クローニングを導入した
マルチエージェントシステムの自己組織化

中井 喜之

内容梗概

大規模な計算能力を得るために，並列コンピューティングや分散コンピューティングが実用化されている．分散問題解決はマルチエージェントシステムの研究の１つのトピックである．他の分散システムに比べたマルチエージェントシステムの特徴は，エージェントの集中管理や同期が無いことである．

マルチエージェントシステムに使用される組織のデザインは，システム全体の性能に重大で定量的な影響を及ぼすことが多くの研究者により示されている．エージェントの組織には，階層型，連合型，連邦型，市場型など様々な組織デザインがあり，それぞれに長所と短所がある．ゆえに，エージェントの組織をあらかじめ設計して固定することもできるが，組織の形を変えて環境に適応した方が良い．

これまでに様々な組織再編の手法が提案されている．例えばエージェントの移動は，過負荷のマシンからそうでないマシンへエージェントが移る手法である．エージェントの分割は過負荷のエージェントが新しいエージェントを作ってタスクの一部を新しいエージェントに割り当てる手法である．エージェントクローニングは本研究が注目した手法であり，過負荷のエージェントが新しいエージェントを作り，自分の能力をコピーして与えるものである．

マルチエージェントシステムにおけるこれらの組織再編プロセスは，いかなる全体の制御も無しに起こることが期待される．クローニングを導入する主たる目的は，タスクに内在するデータ並列性を活用することであるが，以下のような２つの問題が見つかった．

1. これまでのクローニングに関する研究では，各エージェントの担当する処理は実行中に変化せず，エージェント間のデータ依存は非常に単純であった．そして，すべてのエージェントはクローニング可能であるため，クローニングによって効率が良くなるのは当然のことであった．もっと複雑なエージェントのモデルにクローニングを導入すると，より難しくて面白い問題があるかもしれない．
2. そのような複雑なモデルにおいてはエージェントは必ずしもクローニング

可能ではない．我々はそのエージェントがクローニング可能なのかどうか知りたい．もしシステム内にクローンが存在すれば，他のエージェントがデータを送る前に分割するデータと分割しないデータが存在する．それではどのデータは分割すべきで，どのデータは分割すべきでないのだろうか．

これらの問題に対する本研究の貢献は以下の通りである．

1. 既存の自己組織化可能なエージェントのモデルにクローニングを導入した．このモデルでは，エージェントが自身を 2 体のエージェントに分割することや 2 つのエージェントが 1 つに合併することが可能である．クローニングに必要ないくつかの知識をエージェントに追加して，クローニングとデクローニングの詳細なプロセスを提案した．
2. エージェント内部の問題解決のプロセスはデータ依存グラフの形で表すことができる．そのエージェントのデータ依存グラフと近隣のエージェントのグラフのいくつかのノードを解析することで，クローニング可能かどうかが分かる．ゆえに各エージェントは自分がクローニング可能かどうか自身で判断することができる．これを「エージェントのクローニング可能性問題」と呼び，制約充足問題として形式化した．

これらの貢献を評価するため，Scheme で書かれた既存のプロダクションシステムを拡張したシミュレータを開発している．プロダクションシステムは，分散システムでのエージェントによるデータ駆動型問題解決のシミュレーションに適した簡明なモデルである．

現在このシミュレータを使って可能なことは，複数のエージェントの組織による問題解決の過程をシミュレートすることである．自己組織化は未実装のため，エージェントの組織再編は手動で行う必要がある．

このシミュレータを用いて，提案手法によってクローニング可能と分かるエージェントを作り，小さなタスクを与えた．次に，そのエージェントを 2 体に分割して同じ小さなタスクを与えた．最後にはじめのエージェントをクローニングして同じタスクを与えた．シミュレータは予想通りに動き，本研究の主張が有効であることが確かめられた．

本研究の応用領域として，並列計算機の負荷分散あるいは，多数のワークフローを実行する企業のモデル化と再編のシミュレーションが考えられる．

Introducing Cloning to Self-Organization of Multi-agent Systems

Yoshiyuki NAKAI

Abstract

Paralell/Distributed computing has been put into practical use for the purpose of gaining large-scale computational power. Distributed problem solving is a topic of research in multi-agent systems. The characteristics of multi-agent systems, compared to other distributed systems, is that there is no centralized control over agents or synchronization.

Many researchers have showed that the organizational design used by an multi-agent system can have significant, quantitative effect on its overall performance. There are various organizational designs such as hierarchies, coalitions, federations, and markets. Each of these designs has advantages and disadvantages, therefore it is better for agents to reorganize themselves and adapt to their environment than to fix their organization.

A lot of reorganization methods have been proposed so far. Agent migration, for example, is an approach where agents move to less loaded machine from overloded machine, and agent decomposition is another approach where an overloaded agent creates a new agent and move some of its tasks to the new one. Agent cloning which I'm focusing on in this research is an approach where an overloaded agent creates a new agent and copy its ability to the new one.

These reorganization processes in multi-agent systems are also expected to occur without any global control. The main purpose of introducing agent cloning is to exploit data parallelism inherent in tasks, but I found two following problems.

1. In previous research on cloning, tasks for each agents do not change during execution, and data dependencies between agents are very simple. All agents are able to clone, so speeding up with cloning is no wonder. Introducing cloning to more complex agent model could be more difficult and interesting.
2. Agents are not always be able to clone in such a complex model. We want to know which agents are cloneable and which are not. If there exist

clones in a system, other agents should divide some data and not divide other data before sending them to the clones, The question is "which data should agents divide, and which should not?"

My contribution to these problems are as follows.

1. I introduced cloning to an existing model of agent which can be self-organized. In this model, agents are allowed to perform decomposition, dividing onself into two agents, and composition, merging two agents into one. I gave some additional knowledge to agents which is neccesary for cloning, and proposed the process of cloning and decloning in detail.
2. The process of problem solving inside one agent can be expressed in a form of data dependency graph. We can tell whether the agent is cloneable or not, analyzing its data dependency graph and some nodes of neighbor's graph. So each agent can decide whether it is able to clone or not for oneself. I named this problem "Agent Cloneability Problem" and formalize it as a Constraint Satisfaction Problem.

In order to evaluate what I figured out, I'm developing a simulator which extends an existing production system written in Scheme. I chose production system since it is a simple excellent model and especially good at simulating data driven problem solving by agents in distributed systems.

What this simulator can do now is to simulate the process of problem solving by multi-agent organization. I have to reorganize agents for myself since self-organization does not work.

Using this simulator, I made an agent which is cloneable according to my theory, and gave a small task to the agent. Then I divided the agent into two agents and gave the same small task to two cooperative agents. I also made a clone of this agent, and gave divided task to two clones. The simulator worked as I had expexted and I'm sure my idea in this research is effective.

We can apply the knowledge gained from this research topic for load balancing on parallel/distributed computing environment. The knowledge can also be applied for modeling and analysis of workflows in a big business enterprise.

クローニングを導入した
マルチエージェントシステムの自己組織化

目次

第1章　はじめに

大規模な計算能力を得るために，並列コンピューティングや分散コンピューティングが実用化されている．分散問題解決はマルチエージェントシステムの研究の1つのトピックである．他の分散システムに比べたマルチエージェントシステムの特徴は，エージェントの集中管理や同期が無いことである．

マルチエージェントシステムに使用される組織のデザインは，システム全体の性能に重大で定量的な影響を及ぼすことが多くの研究者により示されている．エージェントの組織には，階層型，連合型，連邦型，市場型など様々な組織デザインがあり，それぞれに長所と短所がある [1]．ゆえに，エージェントの組織をあらかじめ設計して固定することもできるが，組織の形を変えて環境に適応した方が良い．

[2] で提案されている組織の自律再編のモデルでは，タスクを処理可能なエージェントが計算資源を独立に持ち，負荷に応じて分割と合併を繰り返す．その結果，エージェントが与えられた並列計算環境に適応した組織を構成することが示されている．

開始時にはエージェントは1体で，タスクを処理するために必要な知識全てを有している．分割の方式は，過負荷のエージェントが新しいエージェントを作り，タスク処理のための知識の一部を移動するというものである．タスク処理のための知識が別の計算資源に配置されることによって，パイプライン並列，タスク並列が実現される．タスク処理のための知識の数が増えないためデータ並列は実現できない．

エージェント組織による負荷分散では以下の3つの並列性を生かす処理により，システムの性能を向上させる．

1. パイプライン並列

   タスクの処理がいくつかの段階に分けられる場合，各段階を別々のエージェントに割り当てれば，1つのタスクが全ての段階を通過するのを待たずとも，次のタスクを1段階遅れで送り込むことができる．

2. タスク並列

   同じデータまたは異なるデータの集合に対して，異なる処理を同時に行うことができる並列性をタスク並列という．並列な処理同士を別のエージェントに割り当てれば，組織全体の処理効率が上がる．

3. データ並列

   同じ処理手順を複数のエージェントに与え，データを分配すればシステム全体として一定時間に処理可能なタスクが増える．

最近の研究では，エージェントの過負荷を解決するための手段としてエージェントのクローニングが提案されている．[3] で提案されているモデルでは，タスクを処理可能なエージェントが過負荷になると，ローカルまたはリモートの計算資源に自身のクローンを作成する．エージェントの分割，合併は行わない．

エージェント内のタスク処理のための知識については議論されていないが，存在を仮定するとクローニングにより，システム内に同じ処理手順が複数存在することになる．異なる種類のエージェント間のインタラクションが議論されていないためパイプライン並列に関しては考慮されていないが，タスク並列とデータ並列が実現されている．

本研究は [2] をベースに，[3] のような最近の研究を取り入れ，パイプライン並列，タスク並列，データ並列の全てを考慮したメカニズムを実現するもので，[2, 3] に比べて汎用性が高い．

本論文の構成は以下の通りである．

第 2 章では，[2] のマルチエージェントシステムのモデルを基に，クローニング導入が可能になるようなエージェントの拡張を提案する．

第 3 章では，前章で拡張したエージェントについて，正しくクローニング可能であるかを判定する問題について議論する．そして，エージェントの持つ知識から制約充足問題に変換して解くことができることを示す．

第 4 章では，[2] のマルチエージェントシステムにおける組織再編の操作にクローニングとデクローニングを追加して，組織再編のプロセスの詳細を述べる．

第 5 章では，単体のプロダクションシステムを拡張して，複数のエージェントによる問題解決をシミュレートできるようになったシミュレータについて述べる．このシミュレータを用いて，典型的なワークフローを持つタスクを想定したシミュレーションを行い，提案方式の定性的な性質を明らかにする．

第 6 章では，本研究の応用分野として，並列計算機環境での資源割り当て，ビジネスプロセスへの資源割り当てについて述べる．

第2章　エージェントの構成

本章では [2] のマルチエージェントシステムにおけるエージェントのモデルを拡張してクローニング可能なモデルを提案する．エージェントはインタプリタ, 領域知識，組織に関する知識から成る．

2.1　インタプリタ

各エージェントの持つインタプリタは，処理手順に基づきデータを参照，変更する．エージェント内での処理は逐次的に実行される．また，インタプリタは近傍のエージェント間でデータの送受信を行う．データの送受信は処理の同期や組織再編の際に発生する．

2.2　領域知識

領域知識は処理手順を格納する長期記憶と，データを格納する短期記憶から構成される．インタプリタは処理手順に基づきデータを参照，変更する．

1 つの処理手順は，条件部と実行部から成る．条件部には参照するデータとそのデータに対する条件が書かれている．実行部にはデータをどのように変更するかが書かれている．

データの存在が条件の場合は＋参照と呼び，データが存在しないことが条件の場合は－参照と呼ぶ．また，データを追加することを＋変更と呼び，データを削除することを－変更と呼ぶ.

2.3　組織に関する知識

他のエージェントと協力して問題解決を行うための知識である．組織全体ではなく，自分の近隣のエージェントに関する知識のみを持つ.

2.3.1　処理同士の関係

1. データ依存情報
   処理 A が変更するデータを処理 B が参照する場合,「処理 B は処理 A に依存する」という．エージェントは自分の担当する処理に関係があるデータ依存情報を持っている．
2. 分割の必要なデータ依存情報
   他のエージェントのクローニングにより，同じ処理を複数のエージェント

が担当している場合，分割が必要になるデータが存在する．処理 A の実行により追加されたデータを，処理 B を担当する複数のエージェントに分割して届ける必要があることを通常のデータ依存情報と区別する．

3. 干渉情報
   処理 A と処理 B を同時に実行すると，どの順序で片方ずつ実行した場合とも異なる結果になる場合「処理 A と処理 B は干渉する」という．

2.3.2　処理手順とエージェントの関係

1. 担当情報
   データ依存情報，干渉情報に現れた処理がどのエージェントにより実行されるかという情報である．「処理 A はエージェント P の担当である」という．
2. クローンリスト
   クローニングにより生じた，自分と同じ処理手順の集合を持つエージェントの集合である．クローン同士でのデクローニングの際に用いる．
3. 分割されているデータ
   クローニングにより生じたクローン間で分割して処理しているデータを記憶している．クローン間で分割されないデータも存在するため，このような区別が必要になる．

2.3.3　エージェントと組織の関係

1. 局所的な統計
   エージェント単体の忙しさを表す．あらかじめ決めておいた時間 P に F 回タスクの処理が実行されている場合，稼働率 $R = F/P$ で定義する．
2. 組織に関する統計
   組織全体の応答時間が全エージェントに放送される．$T_{response}$ を応答時間，$T_{deadline}$ を人間の設定した制限時間とする．

第 3 章　クローニング可能性問題の定式化

エージェントのクローニングではデータ並列処理を目指す．クローン同士ではデータの同期を行わず独立に処理を行い，出力データを集めるとエージェント 1 体の時と同じ結果になることが求められる．

「あるエージェント P がクローニング可能か，そのためにはどのような条件を満たせば良いか」という問題は以下のように制約充足問題として定式化する

ことができる．

3.1　データ依存グラフの導入

クローニング可能性問題を考えるために，[4] に倣いデータ依存グラフを導入する．実行時解析ではなく，エージェント P の長期記憶内の処理手順の集合を解析して生成する．

ノードは処理ノードとデータノードの 2 種類である．システムの持つ処理手順 1 つに対して，処理ノード 1 つを割り当てる．データのクラス 1 つに対してデータノード 1 つを割り当てる．実行可能な処理ノードに対して実行待ちの処理が積み上がり，データノードに対して実際のデータが積み上がる様子を想像すると分かり易い．

エッジは＋参照，－参照，＋変更，－変更を表す 4 種類である．参照はデータノードから処理ノードへの有向エッジで表す．変更は処理ノードからデータノードへの有向エッジで表す．＋と－はエッジに付けるラベルで区別する．

クローニング可能性問題を解くためには，このグラフを隣接エージェントの一部にまで拡大する必要がある．隣接エージェントが持つ処理ノードのうち，エージェント P の持つデータに変更を加えるものをデータ依存グラフに追加する．隣接エージェントの情報は，エージェント P の持つ組織に関する知識から得られる．

こうして得られたグラフの全ノードに通し番号を付けて，$node_i(i = 1, 2, ..., n)$ で表す．

3.2　変数と変域

クローニングにより生じたクローン同士では，クローン以外のエージェントが参照するデータが重複無く網羅されている必要がある．重複無く網羅すべき集合は，クローニング前のエージェント 1 体の場合に生じるデータの集合である．その他のデータノードは，全クローンで同じ状態であることも許される．処理ノードでもデータノードに似た 2 つの状態があり得る．詳細は以下の通りである．

1. データノードの場合
    - クローン同士でこのノードの全データを重複無く網羅している．
    - 全クローンがこのノードの全データを持っている．

2. エージェント内の処理ノードの場合
   - クローン同士でこの処理の全実行を重複無く網羅している．
   - 全クローンでこのノードの全実行が生じる．
3. エージェント外の処理ノードの場合
   - 各クローンにこの処理の実行で生じたデータを漏れや重複無く分配する．
   - 全クローンにこの処理の実行によるデータの変更を全て反映する．

各ノードにおける前者の状態を Divided と呼び，後者の状態を Duplicated と呼ぶ．ノードを変数とし，状態を変域とする．すなわち，各 $node_i$ に対して変数 x_i $(i = 1, 2, ..., n)$ を割り当て，変域を $D_i = \{Divided, Duplicated\}$ とする．

3.3 制約

各 $node_i$ に対して以下のような制約が存在する．x の値が $Divided$ であることを $Divided(x)$ と書く．また，x の値が $Duplicated$ であることを $Duplicated(x)$ と書く．

3.3.1 他のエージェントに参照されているデータノードに対する制約

このノードはエージェントの出力であり，クローン同士の出力するデータが重複無く網羅されている必要があるので，以下の制約が存在する．

$$Divided(x_i)$$

3.3.2 データノードに対する制約

データノードは，処理ノードから変更を受ける可能性があり，それらの処理の実行による影響を受けるため，制約が存在する．$node_i$ を＋変更している処理ノードに対応する変数の集合を P_i とする．$node_i$ を－変更している処理ノードに対応する変数の集合を M_i とする．

1. 「$|P_i| = 0$ かつ $|M_i| = 0$」すなわち「このデータを変更する処理が存在しない」という場合はあり得ない．最初から存在したデータもシステム外の何者かにより変更を受けたのだと考え，その何者かの変更を処理ノードで表現する．
2. 「$|P_i| > 0$ かつ $|M_i| = 0$」すなわち「このデータを＋変更する処理のみが存在する」という場合「それら全処理ノードが Divided で，このデータも Divided である」あるいは「それら全処理ノードが Duplicated で，このデー

タも Duplicated である」ことが求められるので以下の制約で表す.

$$(Divided(x_i) \wedge \forall x_j (\in P_i) Divided(x_j)) \vee (Duplicated(x_i) \wedge \forall x_j (\in P_i) Duplicated(x_j))$$

「一部の処理ノードが Divided で，その他の処理ノードが Duplicated である」場合は重複しているデータとそうでないデータの区別がつかなくなり，正しい結果にならない.

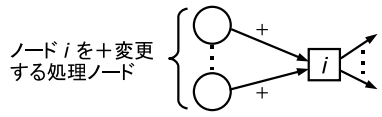

図 1: $|P_i| > 0$ かつ $|M_i| = 0$ であるデータノード i

3. $|P_i| = 0$ かつ $|M_i| > 0$ すなわち「このデータを－変更する処理のみが存在する」という場合「それら全処理ノードが Duplicated であり，このデータノードも Duplicated である」ことが求められるので以下の制約で表す.

$$Duplicated(x_i) \wedge \forall x_j (\in M_i) Duplicated(x_j)$$

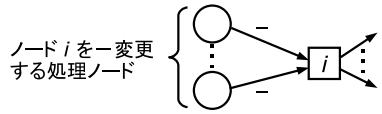

図 2: $|P_i| = 0$ かつ $|M_i| > 0$ であるデータノード i

残念ながらこのデータノードを Divided にできるような，処理ノードの状態は存在しない．例えば Divided であるデータノードに Duplicated な－変更を加えると削除が重複することになる．この制約が多く含まれるようなエージェントはクローニング不可能とは言えないがクローニングの恩恵が少ない.

4. $|P_i| > 0$ かつ $|M_i| > 0$ すなわち「このデータを＋変更する処理と－変更する処理の両方が存在する」という場合「それら全処理ノードが Duplicated であり，このデータデータノードも Duplicated である」ことが求められるので以下の制約で表す.

$$Duplicated(x_i) \wedge \forall x_j (\in P_i) Duplicated(x_j) \wedge \forall x_k (\in M_i) Duplicated(x_k)$$

－変更する処理ノードが存在すると＋変更をする処理ノードに対する制約も厳しくなる．

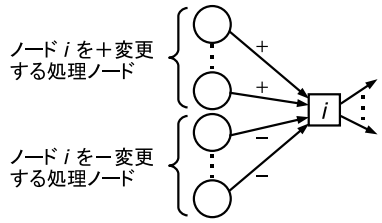

図 3: $|P_i| > 0$ かつ $|M_i| > 0$ であるデータノード i

3.3.3 処理ノードに対する制約

処理は，参照しているデータの有無に影響を受けるため，制約が存在する．$node_i$ が＋参照しているデータノードに対応する変数の集合を P_i とする．$node_i$ が－参照しているデータノードに対応する変数の集合を M_i とする．

1. $|P_i| = 0$ かつ $|M_i| = 0$ すなわち「この処理が参照するデータが存在しない」という場合制約は存在しない．データ依存グラフを隣接エージェントの処理ノードに拡大し，その処理ノードが参照するデータまでは考慮しないのでこのようなノードが存在する．

図 4: $|P_i| = 0$ かつ $|M_i| = 0$ である処理ノード i

2. $|P_i| > 0$ かつ $|M_i| = 0$ すなわち「この処理が＋参照するデータが存在し，－参照するデータが存在しない」という場合「＋参照する全データノードが Duplicated で，この処理ノードも Duplicated である」あるいは「＋参照するデータノードのうち 1 つだけ Divided で他のデータノードは Duplicated であり，この処理ノードは Divided である」ことが求められるので以下の制約で表す．

$$
\begin{aligned}
& (Duplicated(x_i) \wedge \forall x_j (\in P_i) Duplicated(x_j)) \\
\vee \quad & (Divided(x_i) \wedge \exists x_j (\in P_i) Divided(x_j) \wedge \forall x_k (\in P_i / \{x_j\}) Duplicated(x_k))
\end{aligned}
$$

＋参照するデータノードのうち 1 つ以上を Divided にすると，この処理ノー

ドの発火に漏れが生じて正しい結果にならない.
この処理ノードが Divided であるような変数の値の選び方は複数あり，クローニングの効率を上げる工夫の余地がある.

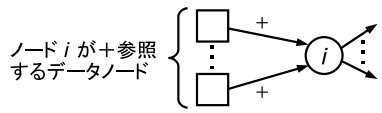

図 5: $|P_i| > 0$ かつ $|M_i| = 0$ である処理ノード i

3. $|P_i| = 0$ かつ $|M_i| > 0$ すなわち「この処理が＋参照するデータが存在せず，－参照するデータが存在する」という場合「－参照するデータノードが全部 Duplicated で，この処理ノードも Duplicated である」ことが求められるので以下の制約を生成する.

$$Duplicated(x_i) \wedge \forall x_j (\in M_i) Duplicated(x_j)$$

－参照されているデータノードは存在しないことが処理実行の引き金になるため，常に Duplicated でなくてはならない.

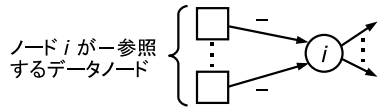

図 6: $|P_i| = 0$ かつ $|M_i| > 0$ である処理ノード i

4. $|P_i| > 0$ かつ $|M_i| > 0$ すなわち「この処理が＋参照するデータと－参照するデータの両方が存在する」という場合「参照する全データノードが Duplicated で，この処理ノードも Duplicated である」あるいは「＋参照するデータノードのうち 1 つだけが Divided で，他のデータノードは Duplicated であり，この処理ノードの発火の重複が無い」ことが求められるので以下の制約で表す.

$$\begin{aligned} & (Duplicated(x_i) \wedge \forall x_j (\in P_i) Duplicated(x_j) \wedge \forall x_k (\in M_i) Duplicated(x_k)) \\ \vee \quad & (Divided(x_i) \wedge \exists x_j (\in P_i) Divided(x_j) \wedge \forall x_k (\in P_i / \{x_j\}) Duplicated(x_k) \\ & \wedge \forall x_l (\in M_i) Duplicated(x_l)) \end{aligned}$$

＋参照されているデータノードのうち 1 つだけが Divided で，それ以外の

データノードは Duplicated でなくてはならない.

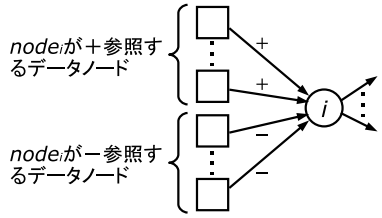

図 7: $|P_i| > 0$ かつ $|M_i| > 0$ である処理ノード i

3.4 解の意味

与えられた制約を満たす組み合わせが存在しない場合，そのエージェントはクローニング不可能である.

与えられた制約を満たす組み合わせ $(x_1, x_2, ..., x_n)$ が存在する場合，そのエージェントはクローニング可能である.

データノードに対応する変数の値が *Divided* ならばクローニングの際にデータを重複無く分割する必要がある．変数の値が *Duplicated* ならばデータを全部コピーする.

エージェント外の処理ノードに対応する変数の値が *Divided* の場合はクローニングの際にその処理を持つエージェントに対してデータの分配をするよう依頼する必要がある．変数の値が *Duplicated* の場合はその処理を持つエージェントから全クローンに対して同じデータが追加または削除される.

3.5 例題

ここまで述べてきた方法でクローニング可能性問題が制約充足問題に変換できることを例題によって確かめる．紙面の都合で変数の値である *Divided* のことを *Div*，*Duplicated* のことを *Dup* と省略表記する.

例 1.

角の丸い長方形は，クローニング可能性判定の対象となるエージェントの範囲である．エージェントはこの角丸長方形内の処理ノードとデータノードを持っている．丸が処理ノードで，四角がデータノードである．各ノード内に書いた数字が変数の添え字に対応するものとする.

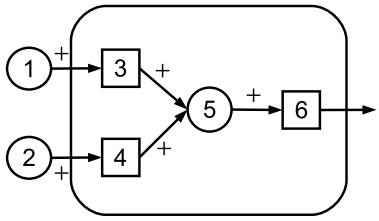

図 8: 例 1

変数：

$$x_1, x_2, x_3, x_4, x_5, x_6$$

制約：

$$\begin{aligned}
C_0(x_6) &= \{(Div)\} \\
C_1(x_5, x_6) &= \{(Div, Div), (Dup, Dup)\} \\
C_2(x_3, x_4, x_5) &= \{(Div, Dup, Div), (Dup, Div, Div), (Dup, Dup, Dup)\} \\
C_3(x_1, x_3) &= \{(Div, Div), (Dup, Dup)\} \\
C_4(x_2, x_4) &= \{(Div, Div), (Dup, Dup)\}
\end{aligned}$$

他のエージェントに参照されているデータノードに対する制約から C_0，データノードに対する制約-2 から C_1，C_3，C_4，処理ノードに対する制約-2 から C_2 が存在する．

解： クローニング可能

$$(x_1, x_2, x_3, x_4, x_5, x_6) = (Dup, Div, Dup, Div, Div, Div), (Div, Dup, Div, Dup, Div, Div)$$

x_3 または x_4 に対応するデータノードのいずれかを隣接エージェントが分割して届ければ x_6 に対応するデータノードはクローン間で Divvided になり正しくクローニングできている．

例 2.

データ依存グラフが閉路を持つ場合もある．

変数：

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8$$

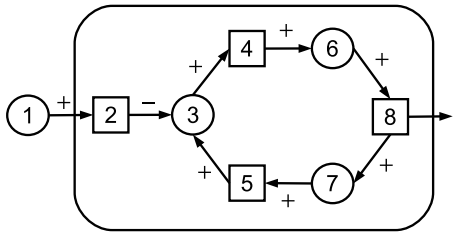

図 9: 例 2

制約：

$$\begin{aligned}
C_0(x_8) &= \{(Div)\} \\
C_1(x_6, x_8) &= \{(Div, Div), (Dup, Dup)\} \\
C_2(x_4, x_6) &= \{(Div, Div), (Dup, Dup)\} \\
C_3(x_3, x_4) &= \{(Div, Div), (Dup, Dup)\} \\
C_4(x_2, x_3, x_5) &= \{(Dup, Div, Div), (Dup, Dup, Dup)\} \\
C_5(x_1, x_2) &= \{(Div, Div), (Dup, Dup)\} \\
C_6(x_5, x_7) &= \{(Div, Div), (Dup, Dup)\} \\
C_7(x_7, x_8) &= \{(Div, Div), (Dup, Dup)\}
\end{aligned}$$

他のエージェントに参照されているデータノードに対する制約から C_0，データノードに対する制約-2 から C_1，C_3，C_5，C_6，処理ノードに対する制約-2 から C_2，C_7，処理ノードに対する制約-4 から C_4 が存在する．

解：クローニング可能

$$(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (Dup, Dup, Div, Div, Div, Div, Div, Div)$$

閉路上のノードを全て Divided にすれば，エージェントへの入力である x_2 に対応するデータノードが Duplicated でも，エージェントの出力である x_8 に対応するデータノードが Divided になり正しくクローニングできている．

例 3.

変数：

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8$$

制約：

$$C_0(x_7) = \{(Div)\}$$

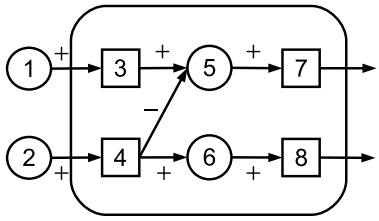

図 10: 例 3

$$
\begin{aligned}
C_1(x_8) &= \{(Div)\} \\
C_2(x_5, x_7) &= \{(Div, Div), (Dup, Dup, \} \\
C_3(x_6, x_8) &= \{(Div, Div), (Dup, Dup, \} \\
C_4(x_3, x_4, x_5) &= \{(Dup, Dup, Dup), (Div, Dup, Div)\} \\
C_5(x_4, x_6) &= \{(Div, Div), (Dup, Dup)\} \\
C_6(x_1, x_3) &= \{(Div, Div), (Dup, Dup)\} \\
C_7(x_2, x_4) &= \{(Div, Div), (Dup, Dup)\}
\end{aligned}
$$

他のエージェントに参照されているデータノードに対する制約から C_0，C_1，データノードに対する制約-2 から C_2，C_3，C_6，C_7，処理ノードに対する制約-4 から C_4，処理ノードに対する制約-2 から C_5 が存在する．

解：クローニング不可能

x_8 に対応するノードを Divided にするために x_4 に対応するノードを Divided にしたいのであるが，制約の C_4 によりそれはできない．

第 4 章　エージェントの組織再編のための基本操作

エージェントの組織再編のルールと，組織再編で用いられる基本操作について，詳細を述べる．この章も第 2 章と同じで [2] のモデルを拡張したものである．過負荷ならば，エージェントは分割またはクローニングを行う．負荷が少なければ，エージェントは合併またはデクローニングを行う．負荷は第 2 章で定義した稼働率 R，応答時間 $T_{response}$，制限時間 $T_{deadline}$ を用いて判断する．

4.1　組織再編のルール

1. 「$2R < 1$」ならば，稼働率が低いので他のエージェントとの合併を試みる．クローンが存在する場合はデクローニングを試みる．合併やデクローニン

グは 2 体のエージェントが合意した場合にのみ実行される．

2. 「$T_{deadline} > T_{response}$ かつ $2R < T_{deadline}/T_{response}$」ならば，組織全体の応答時間が制限時間に比べて十分に短いので，他のエージェントとの合併またはデクローニングを試みる．
3. 「$T_{deadline} < T_{response}$ かつ $R = 1.0$」ならば，組織は制限時間を守っておらず，このエージェントはビジーである．エージェントがクローニング可能か調べ，クローニング可能ならばクローニングを行い，不可能ならば分割を行う．

4.2 分割

過負荷に対してはクローニングを試み，不可能であれば分割を行う．以下ではエージェント P をエージェント Q に分割する手順を示す．

1. エージェントの生成
   エージェント Q を生成する．エージェント Q は直ちに処理を実行し始める．
2. 移動する処理手順の決定
   エージェント P からエージェント Q に移動する処理手順を決定する．このモデルでは，エージェントの持つ処理手順の集合を任意に 2 分割して，その片方を P から Q に移動する．
3. 同期依頼
   移動を予定している処理手順について，同期依頼をエージェント Q に送る．また，隣接エージェントには移動予定の処理手順とデータ依存である，あるいは干渉する処理を実行しないように同期依頼を送る．全ての同期依頼に対して確認メッセージを受け取ったら，次のステップに進む．
4. 処理手順の移動
   エージェント Q に処理手順を移動する．また，移動する処理手順が参照するデータをコピーする．エージェント P から不要になったデータを削除する．
5. データ依存，干渉情報の移動
   移動された処理を含むデータ依存，干渉情報をエージェント Q にコピーする．エージェント P からは不要になった情報を削除する．隣接エージェントには，処理手順の移動を報告する．通知を受けたエージェントは処理手順の担当情報を書き換える．
6. 同期解除

エージェントQと隣接エージェントに同期解除メッセージを送信し，同期を解除する．

4.3 合併

エージェントにクローンが存在しない場合は，合併により計算資源を解放することができる．以下ではエージェントPを隣接エージェントに合併する手順を示す．

1. 合併依頼
   隣接エージェントに合併依頼メッセージを送信する．あるエージェントQが合併を承認したとする．合併を承認するエージェントが無ければ合併は行われない．
2. 同期依頼
   エージェントPの持つ処理手順全てについて，同期依頼をエージェントQに送る．また，隣接エージェントにはエージェントPの持つ処理手順とデータ依存，あるいは干渉する処理を実行しないように同期依頼を送る．全ての同期依頼に対して確認メッセージを受け取ったら，次のステップに進む．
3. 処理手順とデータの移動
   エージェントQに処理手順をコピーする．また，処理手順が参照するデータをコピーする．
4. データ依存，干渉情報の移動
   移動された処理手順を含むデータ依存，干渉情報をエージェントQにコピーする．エージェントPからは不要になった情報を削除する．隣接エージェントには，処理手順の移動を報告する．通知を受けたエージェントは処理手順の担当情報を書き換える．
5. 同期解除
   エージェントQと隣接エージェントに同期解除メッセージを送信し，同期を解除する．
6. エージェントPは消滅する．

4.4 クローニング

過負荷に対して，まずエージェントのクローニングを試みる．エージェントPをエージェントQにクローニングする手順を述べる．

1. クローニング可能性の判定
   第3章の手法に従って，エージェントPがクローニング可能であるかを判定する．可能である場合は次のステップに進む.
2. エージェントの生成
   エージェントQを生成する．エージェントQは直ちに処理を実行し始める.
3. 同期依頼
   エージェント内の全処理手順について，同期依頼をエージェントQに送る．また，隣接エージェントにはエージェントPの持つ処理手順とデータ依存，あるいは干渉する処理を実行しないように同期依頼を送る．全ての同期依頼に対して確認メッセージを受け取ったら，次のステップに進む.
4. 処理手順とデータのコピー
   エージェントQに全処理手順をコピーする．クローン間で分割しないデータについては全部コピーする．クローン間で分割するデータは半分を移動する.
5. データ依存，干渉情報の移動
   全てのデータ依存，干渉情報をエージェントQにコピーする．隣接エージェントには，処理手順のコピー報告とデータの分割依頼を送信する．処理手順のコピー報告を受けた隣接エージェントはコピーされた処理手順の担当情報を追加する．データの分割依頼を受けた隣接エージェントは，該当するデータ依存情報を削除して，分割が必要なデータ依存情報に差し替える.

4.5 デクローニング

クローニングを行ったエージェントおよびクローニングにより生じたエージェントは，負荷が減ってもクローン以外のエージェントと合併してはいけない．クローン同士の合併をデクローニングと呼ぶ.

以下ではエージェントPをデクローニングする手順を述べる.

1. デクローニング依頼
   クローンであるエージェントにデクローニング依頼メッセージを送信する．あるエージェントQがデクローニングを承認したとする．デクローニングを承認するエージェントが無ければデクローニングは行われない.
2. 同期依頼
   エージェントPの持つ全処理手順について，同期依頼をエージェントQに

送る．また，隣接エージェントにはエージェントＰの持つ処理手順とデータ依存，あるいは干渉する処理を実行しないように同期依頼を送る．全ての同期依頼に対して確認メッセージを受け取ったら，次のステップに進む．

3. データの移動
   エージェントＰの持つデータのうち，クローン間で分割されていたデータをエージェントＱに移動する．
4. データ依存，干渉情報の移動
   隣接エージェントにエージェントＰの消滅を予告する．通知を受けたエージェントはエージェントＰに関する処理手順の担当情報を削除する．
5. 同期解除
   エージェントＱと隣接エージェントに同期解除メッセージを送信し，同期を解除する．
6. エージェントＰは消滅する．

第５章　実装と評価

マルチエージェントシステムによる問題解決では，システム全体の制御が存在せず，エージェント同士のデータ交換が処理実行の引き金になる．プロダクションシステムはこのようなデータ駆動型の問題解決のシミュレーションに適した簡明なモデルである．

そこで単体のプロダクションシステムを拡張し，複数のエージェントによる問題解決のシミュレーションを可能にした．エージェントの組織再編はまだ実装していない．組織の編成を提案手法に従って手動で行ってから，シミュレータ上でタスクの処理を試した．

このシミュレーションにおいて，提案手法に基づくクローニング結果が正しいことを確かめた．また，分割よりもクローニングが有効である場合が存在することを確かめた．

以下では，プロダクションシステムの用語に合わせ，処理手順のことをルールと呼ぶ．処理の実行はルールの発火と言う．

5.1 プロダクションシステムを用いたシミュレータの実装

元となるプロダクションシステムは，ルールを格納するプロダクションメモリ，ルールの実行に必要なデータを格納するワーキングメモリ，双方のメモリを参照し解釈実行するインタプリタから構成されている．

このプロダクションシステムで複数のエージェントによる問題解決を擬似的に実現するため，以下の拡張を行った．

1. ワーキングメモリに格納されるデータにメタ情報としてエージェント名を付けられるようにした．エージェント名が同じであるデータの組み合わせのみ参照されるようにした．
2. 各エージェントが持つルールを定義できるようにした．
3. 1 サイクルでルールを発火可能な全エージェントがルールを 1 つずつ発火できるようにした．
4. データの変更が 1 サイクルの遅延の後に，隣接エージェントに反映されるようにした．遅延のサイクル数は可変な実装になっている．
5. 同時に発火できないルールの組み合わせをあらかじめ決めておけば，片方のルールが発火したときには，もう一方のルールをデータ変更の遅延サイクルと同じサイクル数発火できないようにした．これによりルール同士の干渉を防ぐことができる．

5.2 動作確認

典型的な例題をシミュレータ上で動作させた．

例 1. エージェント 1 体

エージェント P が図 11 のデータ依存グラフで表すことができるとする．データのクラスには D1，D2，D3，D4，D5 と名前をつけ，ルールには R1，R2 と名前をつけた．D1，D2，D3 への＋変更は外部からエージェントへの入力を意味する．最終的なエージェントの出力は D5 である．

本シミュレータでは図 12 のように `defrule` でルールを定義し，`defagent` でエージェントを定義できる．ルールの定義では`-->`の前にルールの発火条件を書き，後にデータの変更を書く．エージェントの定義ではエージェントの持つエージェントの名前とエージェントの持つルールを書く．

このエージェント P に D1，D2，D3 を 2 つずつ与える．D1，D2 の組み合わ

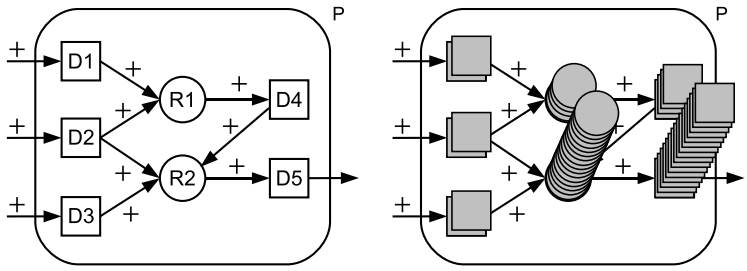

図 11: エージェント P のデータ依存グラフと処理実行のイメージ

```
(defrule R1      (defrule R2      (defagent agentP
  (D1)             (D2)             (R1 R2))
  (D2)             (D3)
  -->              (D4)
  (make (D4)))     -->
                   (make (D5)))
```

図 12: ルールとエージェントの定義

せにより D4 が 4 つ生成され，D2，D3，D4 の組み合わせにより D5 が 16 個生成されれば良い．

実際にシミュレータで実行すると，1 サイクルで D4 が生成され，次の 4 サイクルで D5 が 4 個生成された．この 5 サイクルが 4 回繰り返され 20 サイクルで処理が終了した．D5 は 16 個生成された．

例 2. 分割されたエージェント 2 体

図 13 のように，例 1 のエージェント P を分割してエージェント P と Q に分けた．エージェント間のデータ移動のコストは 1 サイクルとした．エージェント 1 体の場合と比較するために，エージェント P には D1 と D2 を 2 つずつ与え，エージェント Q には D2 と D3 を 2 つずつ与えた．エージェント P とエージェント Q に与えた D2 は同じものである．

1 サイクル目：エージェント P に D4 が生成された．2 サイクル目：エージェント P で D4 がもう 1 つ生成され，エージェント Q に 1 サイクル目で生成された D4 が届いた．3 サイクル目：エージェント P で D4 が生成された．エージェント Q には D4 がもう 1 つ届いて，D5 が生成された．4 サイクル目：3 サイクル目と同じ．

5 サイクル目：エージェント Q には最後の D4 が届き，D5 が生成された．これ以降エージェント P は仕事をしないのでエージェント Q に合併されたと考え

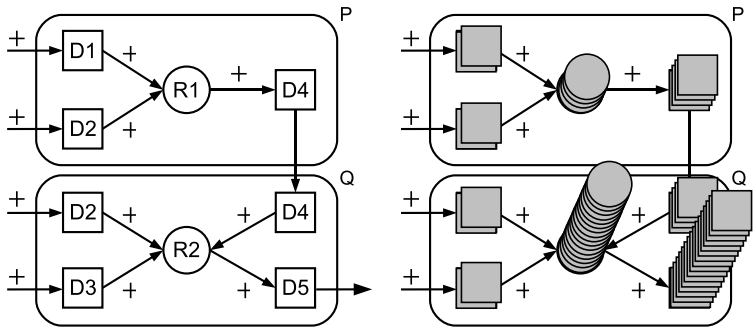

図 13: エージェント P と Q のデータ依存グラフと処理実行のイメージ

る．6 から 18 サイクル目：エージェント Q では 13 個の D5 が生成されて終了した．

5 サイクル目でエージェント P がエージェント Q に合併された後は，エージェント 1 体で処理をするため，はじめからエージェント 1 体の場合に比べてあまり速くならない．

例 3. クローニングされたエージェント 2 体

例 1 のエージェント P は第 3 章で提案したクローニング可能性を求める問題を解くとクローニング可能であると分かる．図 14 のように，D1 を P と P' に分けて届ければ，それぞれが出力する D5 を集めた結果がエージェント 1 体のときと同じになる．

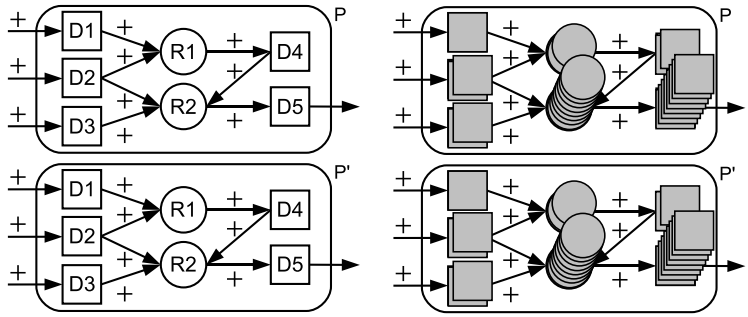

図 14: エージェント P と P' のデータ依存グラフと処理実行のイメージ

サイクル 1：エージェント P とエージェント P' の両方で D4 が生成された．サイクル 2 から 5：エージェント P とエージェント P' の両方で D5 が生成された．

この 5 サイクルが 2 回繰り返されて，最終的に D5 は合わせて 16 個になった．サイクル数は 10 回でエージェント 1 体の時の 20 回に比べて約半分になった．は

じめから終わりまで2体のエージェントが仕事を続けた．

正確にはデータの個数だけでは，クローニングの正しさは分からない．本シミュレータの実装により，どのデータの組み合わせで，どのデータが生じたのか追跡可能となった．

第6章　応用

本研究の応用領域として，並列計算機の負荷分散に加えて企業情報システムが対象としているような多数のワークフローを実行する企業のモデル化とその再編のシミュレーションが考えられる．

6.1　並列計算環境での資源割り当て

単体のプロセッサの処理速度向上に限界が見え始めた近年，マルチコアプロセッサの開発に加えて，大規模クラスタやインターネット上での分散コンピューティングが注目され，大規模並列計算環境が共有される機会が増えている．そこで，ある並列計算環境において利用者が複数いる場合，誰のタスクにどれぐらいの計算資源を割り当てるということが問題になる．

本研究の提案したモデルでは，タスクを処理するために複数のエージェントによる組織が構成される．制限時間内にタスクを処理できていないと忙しいエージェントが分割して組織の再編が行われる．エージェントが増えると，使用する計算資源は増える．逆に制限時間に余裕があればエージェント同士を合併して，計算資源を解放する．

異なる利用者によるタスクは，異なるエージェントの組織により処理される．そこで，複数の利用者間での利害を調整するためには，エージェントの組織間での交渉が必要になる．

例えば2人の利用者Aと利用者Bがいて，利用者Aの利用が優先の場合は利用者Aのエージェントの組織が制限時間を満たすまで利用者Bのエージェントの組織の制限時間を延ばす．制限時間が延びた利用者Bのエージェントの組織は計算資源を解放して利用者Aのエージェントの組織が使える計算資源が増える．

このような交渉により，利用者Bの資源利用が完全に止められること無く，利用者Aの利用が優先される．利用権の定め方とには様々な方式が考えられる．

いかなる方式であっても与えられた計算資源の範囲内で最大限の結果を得るには，柔軟なエージェント組織によるタスク処理が有効である．

6.2　ビジネスプロセスへの資源割り当て

通信網と物流網の世界規模での標準化と普及により，個人や組織が地球上のあらゆる場所から共同作業に参加できるようになった．企業活動はモジュール化され，アウトソーシングと自動化が進んでいる．モジュール化された企業活動を効率的に管理するためにワークフロー・システムの導入が広がっている．ワークフローは再利用が可能で，仕事の担当者が短期間で変わるような状況で有用である．また，ワークフローの分析によりビジネスプロセスの改善を行うことができる．

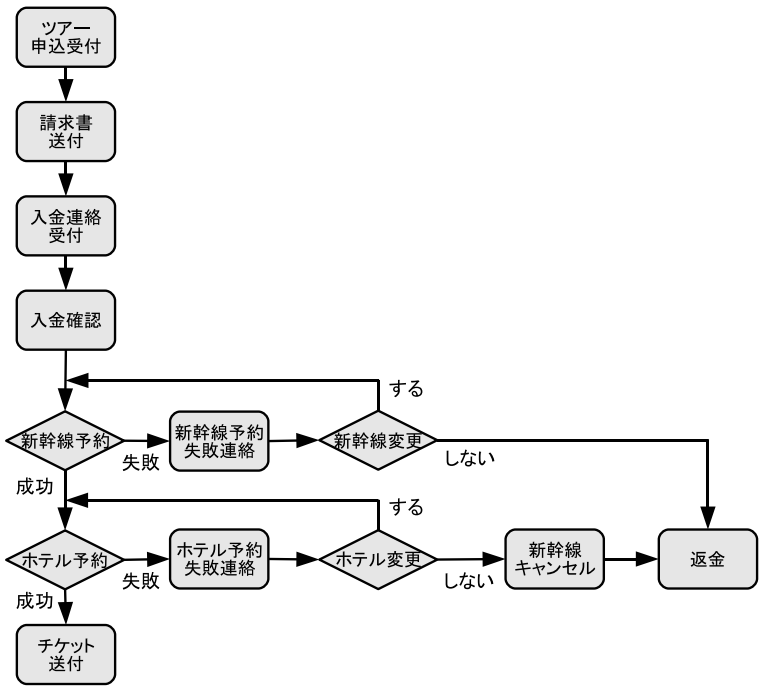

図 15: 新幹線とホテルがセットのツアー予約を処理する旅行会社のワークフロー

ワークフローは処理の内容や順番を流れ図にまとめることが多い．図 15 は新幹線とホテルがセットのツアー予約を処理する旅行会社のワークフローである．角の丸い四角形でビジネスプロセスの手順を表し，菱形で条件分岐を表している．

ツアーの申し込みを受けたら請求書を送り，入金を確認できたら新幹線とホ

テルの予約を行い，成功すればチケットを送付して終了する．途中で失敗すれば内容を変更してやり直すか，返金して終了する．

ワークフローではデータを明示的に書く場合と書かない場合がある．いずれにしても，何らかのデータが生成されたりイベントが発生したりすることで処理が始まる．イベントもデータの一種であると考えると，このビジネスプロセスはデータ依存グラフを用いて表すことができる．

図 15 の例であれば，図 16 のように表すことができる．四角のノードがデータあるいはイベントを表し，丸のノードが処理を表す．エッジはすべて＋で，データの＋参照と＋変更のみで構成されている．

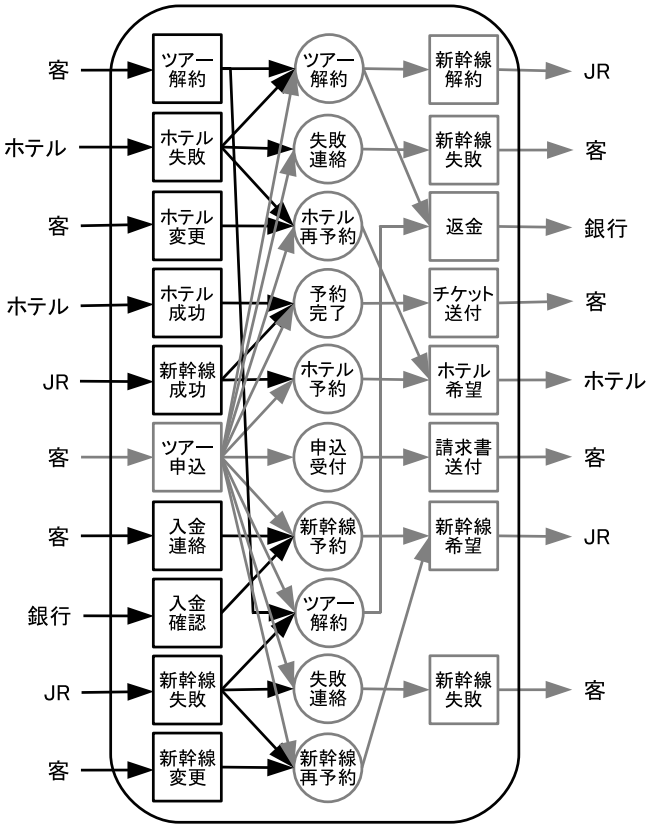

図 16: 客ごとに担当を決める場合のデータ依存グラフ

図 16 の左側が入力で，右側が出力である．処理の流れは図 15 と同じであるが，各処理において必要なデータと生成されるデータが明確になる．

データ依存グラフを本研究の手法に基づき解析すると，灰色で区別したノードを Divided にすればクローニングできることが分かる．ツアー申込の段階で

客ごとに担当するクローンを決めれば，その後生成される全データを共有することで出力は正しく分割される．つまり，ある客に対する担当者が決まっていてツアー予約の全過程を行っている．

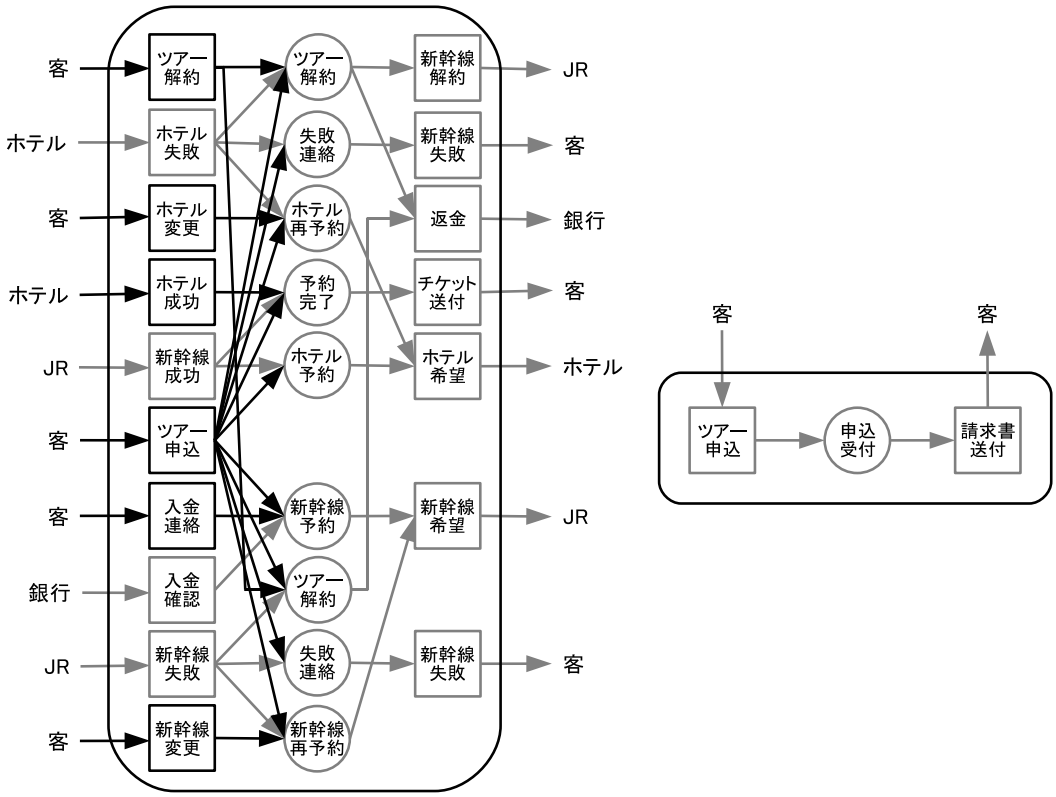

図 17: 申し込み受付を分離する場合のデータ依存グラフ

次に図 16 から申込受付の処理を分離する．請求書送付が重複しないようにツアー申込を分割せざるを得なかったが，この処理を分離することで図 17 のようにより多種のデータを Divided にしても正しく仕事を行うことができる．

右側の受付担当の各クローンは客ごとに対応するが，左側の予約処理担当の各クローンは全ての客の申込を共有している．受付担当と予約処理担当は独立にクローニング可能である．このような処理を人間が行うと混乱するかもしれないが，計算機で自動化する場合はより多種のデータが分割されている方が効率が良い．

グラフの解析により以上のような議論が可能になる．今後は人的資源や計算機資源の自動的な割り当てを検討したい．

第7章　おわりに

本研究では，マルチエージェントシステムによる負荷分散における以下の2つの課題に取り組んだ．

1. これまでのクローニングに関する研究では，各エージェントの担当する処理は実行中に変化せず，エージェント間のデータ依存は非常に単純であった．そして，すべてのエージェントはクローニング可能であるため，クローニングによって効率が良くなるのは当然のことであった．もっと複雑なエージェントのモデルにクローニングを導入すると，より難しくて面白い問題があるかもしれない．
2. そのような複雑なモデルにおいてはエージェントは必ずしもクローニング可能ではない．我々はそのエージェントがクローニング可能なのかどうか知りたい．もしシステム内にクローンが存在すれば，他のエージェントがデータを送る前に分割するデータと分割しないデータが存在する．それではどのデータは分割すべきで，どのデータは分割すべきでないのだろうか．

これらの問題に対する本研究の貢献は以下の通りである．

1. 既存の自己組織化可能なエージェントのモデルにクローニングを導入した．このモデルでは，エージェントは自身を2体のエージェントに分割することや2つのエージェントを1つに合併することが可能である．クローニングに必要ないくつかの知識をエージェントに追加して，クローニングとデクローニングの詳細なプロセスを提案した．
2. エージェント内部の問題解決のプロセスはデータ依存グラフの形で表すことができる．そのエージェントのデータ依存グラフと近隣のエージェントのグラフのいくつかのノードを解析することで，そのエージェントがクローニング可能か可能でないかが分かる．ゆえに各エージェントは自分がクローニング可能かどうか自身で判断することができる．これを「エージェントのクローニング可能性問題」と呼び，制約充足問題として形式化した．

これらの貢献を評価するため，Schemeで書かれた既存のプロダクションシステムを拡張したシミュレータを開発している．プロダクションシステムは，分散システムでのエージェントによるデータ駆動型問題解決のシミュレーションに適した簡明なモデルである．

現在このシミュレータを使って可能なことは，複数のエージェントの組織による問題解決の過程をシミュレートすることである．自己組織化は未実装のため，エージェントの組織再編は手動で行う必要がある．

このシミュレータを用いて，提案手法によってクローニング可能と分かるエージェントを作り，小さなタスクを与えた．次に，そのエージェントを 2 体に分割して同じ小さなタスクを与えた．最後にはじめのエージェントをクローニングして同じタスクを与えた．シミュレータは予想通りに動き，本研究の主張が有効であることが確かめられた．

本研究の応用領域として，並列計算機の負荷分散あるいは，多数のワークフローを実行する企業のモデル化と再編のシミュレーションが考えられる．

謝辞

本研究を進めるにあたり,熱心な支援と適切な指導を賜りました石田亨教授に厚く御礼申し上げます．また，有益な助言を与えてくださいました松原繁夫准教授をはじめ，石田研究室の皆様方に心より感謝いたします．

参考文献

[1] Horling, B. and Lesser, V.: A survey of multi-agent organizational paradigms, *The Knowledge Engineering Review*, Vol. 19–4, pp. 281–316 (2005).

[2] Ishida, T., Gasser, L. and Yokoo, M.: Organization Self-Design of Distributed Production Systems, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4–2, pp. 123–134 (1992).

[3] Shehory, O., Sycara, K., Chalasani, P. and Jha, S.: Agent Cloning: An Approach to Agent Mobility and Resource Allocation, *IEEE Communication Magazine*, Vol. 36–7, pp. 58–67 (1998).

[4] Ishida, T.: Parallel Firing of Production System Programs, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 3–1, pp. 11–17 (1991).