階層的マルチエージェントによる効率的なエージェント間通信

國頭吾郎[†] 吉川典史[†,☆] 相澤清晴[†]

本論文では階層的に配置したエージェントが，ネットワークの中間ノードにおいて何らかの情報処理を行う階層的マルチエージェントシステムを提案する．提案する階層的なエージェント通信により，多対一のトラヒックの集中が生じるシステムにおいて効率的な通信が可能となる．中継処理を行うエージェントの配置法についての検討を行い，システムを動的に再構成する方法を示した．実装実験により提案手法の有効性を示した．

Hierarchical Agents for Efficient Agent Communication

Goro Kunito,[†] Norifumi Kikkawa[†,☆] and Kiyoharu Aizawa[†]

We propose a hierarchical multi-agent system, in which intermediate agents in the hierarchical communication node efficiently collect, process and transmit data between agents. It achieves efficient one-to-many communication between agents. We also discuss allocation of the intermediate agents for dynamic optimization. We examined its efficiency by implementation.

1. まえがき：マルチエージェントシステム

ネットワーク内に存在し，ユーザの代理となって自律的にはたらくプロセスを総称して，ネットワークエージェント，なかでもネットワーク上を移動しながらリモートで仕事を行うものをモバイルエージェントと呼び，様々な研究が行われている[1)~5)]．また，この中で特に複数のエージェントを協調させ，目的を達成させるものをマルチエージェントシステムという．マルチエージェントシステムについていろいろなシステムが考案されており[6)~11)]，これらは大きく次の2つにわけられる．すなわち，

- 自律分散マルチエージェントシステム
- 集中制御型マルチエージェントシステム

である（**図1**）．

自律分散マルチエージェントシステムとは，

- 分散環境との自律的相互作用
- 競合作用を解消するための協調能力
- 経験の学習能力

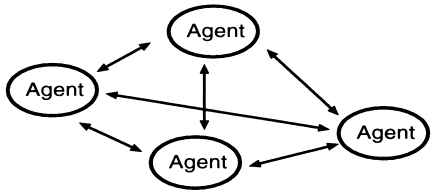
から成るエージェントの集合によって，大域的な目標を達成していく分散処理アーキテクチャである．エージェントは自ら環境の変化を検出し，自主的に行為を決定する自律性と経験の学習能力が必要である．

文献7)では，マルチエージェントを応用して複数の人や組織の間で相互に依存する計画を交渉によって立案/修正する業務の支援システムが開発されている．複数建設所間でのコンクリート塊などの資材融通計画問題である建設副産物リサイクル調整業務を対象に，マルチエージェント技術を適用しているものである（**図2**）．

営業エージェントはネットワークで資材の受入先を探し回り，調達エージェントは受入可能資材情報を提供する．計画エージェントは収集された情報から融通計画案を作成し，許諾請求を調達エージェントに出す．その回答により再立案を重ね，計画を決定する．

一方，エージェントのそれぞれが自律的な機能を持って，完全に機能分散する方法とは異なり，ある程度エージェント群に管理/被管理の構造を用いて機能の集中を図る方が便利な場合がある．たとえば，ネットワークの広域に分散したエージェントからの報告を1カ所に集め，逐次処理して判断を下していくようなシステムでは，分散したエージェントは単に中央の制御エージェントに向けた通信を行い，複雑な機能を持たない．この際分散したエージェント数が増大すると，中央の制御エージェント付近ではトラヒックが増加し，

† 東京大学大学院工学系研究科，新領域創成科学研究科
School of Engineering & School of Frontier Science, University of Tokyo
☆ 現在，ソニー株式会社
Presently with Sony Corporation



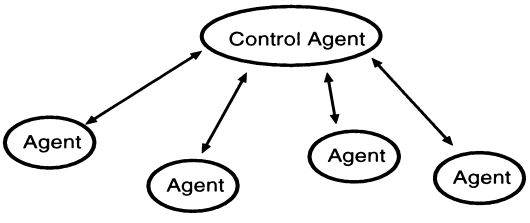


図 1 マルチエージェントシステム
Fig. 1 Multi-agent system.

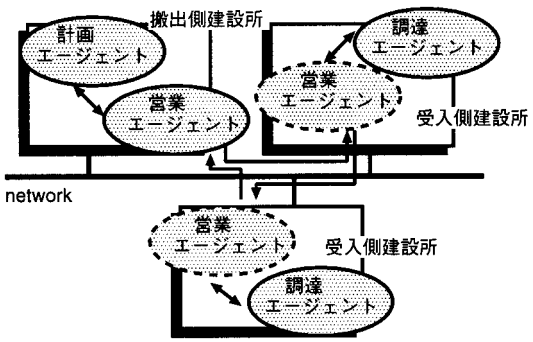


図 2 リサイクル調整支援へのマルチエージェント技術の適用
Fig. 2 An application of a multi-agent system for assistance for recycling systems.

通信路の輻輳や制御エージェントへの負荷集中を引き起こす.

エージェントという用語の使い方が本論文とは異なるものの，文献 11) では，この問題を移動エージェントによるデータ集約という方法で解決している（**図 3**）. Artemis は数万人規模のリアルタイム・データ集約を可能にする同時集約型インタラクティブサービスシステムを目指すものである．インターネット上にエージェントの動作環境を持つ中継ノードを用意し，その中継ノード上で参加者からのデータを運んできたモバイルエージェントがデータの中継集約処理を行いながらイベントサーバへと集約させる．特徴として，中継ノードでのプログラムは中継ノードに置くのではなく，各エージェント内に定義している．これによって様々なサービスに対応して中継ノードを再設定する必要がない反面，同じプログラムが何度もネットワークを移動することによるロスが発生する．ただし，実際使用されるときには任意のサービスが同時に実行されることはあまりなく，数種の専門的なサービスが定義されたインテリジェントノードの方が効率は良いものと考えられる.

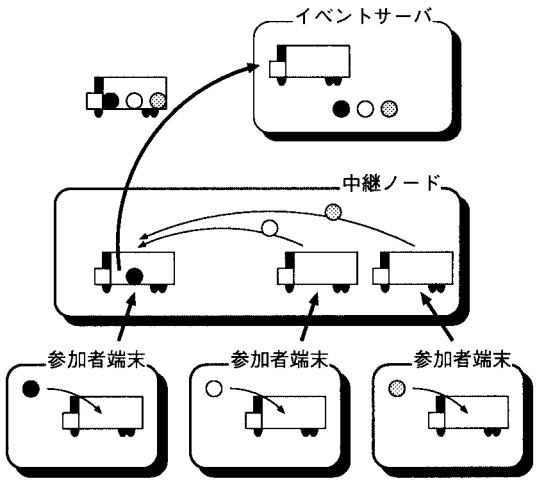


図 3 移動エージェントを用いたデータ集約システム
Fig. 3 A mobile agent system for summarizing data.

以上のマルチエージェントの制御方式に対し，本論文では階層性を持たせた集中制御方式を提案する[12]. エージェントの多対一の双方向通信において，中間に情報集約処理を行うエージェントを有する階層的な構造により，通信効率が大きく向上する．本提案は 7 章で示す，実時間で大量のアクセスのあるような用途で有効と考えられる.

以下，2 章では階層的マルチエージェントとその利点を論じ，3 章で情報集約を行うエージェントの役割と配置法について述べる．4 章では実装について，5 章，6 章にてその実装による実験をまとめる．7 章にて本提案のシステムの応用に触れ，8 章でまとめる.

2. 階層的マルチエージェントシステム

本論文では，集中制御型マルチエージェントシステムにおけるエージェント間の多対一の双方向通信を対象とする．多数ある方を Slave Agent とし，これら Slave Agent は 1 つの Central Agent に対して通信するものとする．それぞれの Slave Agent は機能の限られたリアクティブ型のエージェントで，Central Agent は熟考型エージェントであるとする．Central Agent が Slave Agent に対して指令などを送ると，各 Slave Agent から何らかの処理結果が送られるようなシステムを考える．ここで Slave Agent の数が非常に

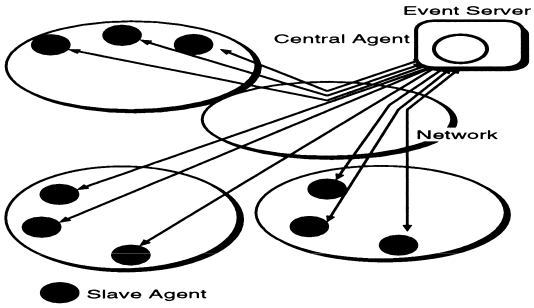


図 4 集中制御型マルチエージェントシステム
Fig. 4 A centrally controlled multiagent system.

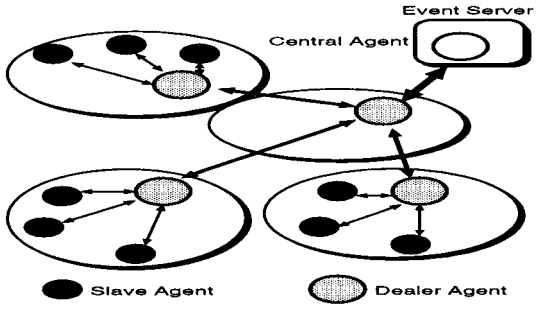


図 5 階層的マルチエージェントシステム
Fig. 5 A hierarchical multiagent system.

多い場合，Central Agent 周辺のネットワークにかかる負荷は大きく，また Central Agent の存在するホストへの負担も大きい（**図 4**）．

そこで，本論文では**図 5** に示すようにエージェント間通信を階層化することを提案する．末端の Slave Agent をいくつかのグループに分け，その 1 つのグループを制御する新たなエージェントとして Slave Agent よりは知的な Dealer Agent を導入する．この階層化は複数階層にわたるものとする．

Dealer Agent は上位の Dealer Agent または Central Agent よりも下位の Dealer Agent もしくは Slave Agent に近いところに配置し，下位の Dealer Agent および Slave Agent の位置情報を管理する．

下位のエージェントから上位へ送るデータについては，メッセージの集約やアプリケーションに依存する統計処理や中間処理を Dealer Agent で行うことができ，その結果を上位の Dealer Agent に送る．Central Agent から Slave Agent へのメッセージは Dealer Agent を経由し，各 Dealer Agent でメッセージが複製されて下位エージェントそれぞれに送られる．

システムを階層的にすることによって次のような利点があげられる．

- 負荷分散
  負荷の集中は，Central Agent での処理が 1 つのホストで行われなければならないところに起因している．階層的マルチエージェントシステムでは，Central Agent での処理の一部を Central Agent と Slave Agent の間の通信経路の途中のホストに配置した Dealer Agent が代行する．このことによって Central Agent の存在するホスト，またその付近のネットワークへの負荷集中を避けることができる．
- 局所的変化の影響の分散
  集中型マルチエージェントシステムではシステムの局所的な状態変化が直接 Central Agent に影響する．たとえば，Central Agent が Slave Agent との通信を維持するために位置情報などを直接管理しているものとする．もしある 1 つの Slave Agent が頻繁に移動するような場合，Central Agent は位置情報の更新処理をそのたびに行わなければならない．階層的マルチエージェントシステムでは，そのような局所的状態変化をその部分の Dealer Agent で吸収することができる．
  また，モバイルホストのように貧弱な通信路の先に Slave Agent が存在するような場合は，通信そのものの負荷を Dealer Agent で吸収できることも大きな特徴である．

3. Dealer Agent による階層的エージェント間通信

新たに導入した Dealer Agent はインテリジェントな処理が可能な中継ノードという側面と，もう 1 つ自らの下位のエージェントを管理するという側面の 2 つの役割を持つ．

3.1 メッセージの中継

Dealer Agent は，下位のエージェントから送られたメッセージを上位へ，上位のエージェントから送られたメッセージを下位へ中継する．ネットワークが混んできた場合，下位から上位へ送られるメッセージは宛名などのヘッダ情報を共有できる部分が多いので，メッセージをいったん Dealer Agent で集積して 1 つのメッセージ群にして上位へ送る．エージェント間通信のようなメッセージ通信の場合，ヘッダのオーバヘッドが大きいためこのような Dealer Agent によるまとめ送りが有効である場合が多い．また，この際即時性の強くないメッセージはそこで Dealer Agent のバッファに貯えられ，即時性の強いメッセージのみを送る．この際貯えられた即時性の強くないメッセージは Dealer Agent において中間処理が施され，ある程度まとまった形にすることができる．通信量が大きくなってきた

場合，Dealer Agent を多段にすることによって通信量の削減/負荷の分散化を図ることができる．また，上位から下位へのメッセージは Dealer Agent 上で下位エージェントの数だけ複製してマルチキャスト型で通信する．

3.2 下位エージェントの管理

モバイルエージェントを扱う際，エージェントの位置情報の管理は重要かつ困難な問題である．この階層的マルチエージェントシステムでは，Central Agent を除くすべてのエージェントは必ず 1 つの上位エージェント（Dealer Agent）を持ち，Dealer Agent は自分の子エージェントの位置情報や名前の管理を行う．

エージェントが移動するときには，まず所属する上位の Dealer Agent にその旨と移動先を知らせ，移動が完了するまでの間，自分に届くデータを保持していてもらう．移動が完了したら再び上位エージェントにそれを知らせ，預ってもらっていたデータを受けとる．場合によっては所属する上位エージェントを変更してもよい．このため処理中のシステムを止めることなくつねにネットワークの状態に即して階層構造を最適化することを図ることができる．これは末端のエージェントが移動する場合だけではなく，Dealer Agent の移動に応用することも可能である．

エージェントの管理，および移動の詳細については 4.1，4.2 節で述べる．

3.3 階層構造の動的再構成

メッセージに大きさ W のデータを載せて通信を行う場合，そのときの遅延 D は定数 α，D_β を用いて $D \simeq \alpha W + D_\beta$ のように表せる．このとき W の小さなところでは $\alpha W \ll D_\beta$ であり，

$$D \simeq D_\beta \tag{1}$$

この場合，通信遅延はメッセージ数のみにほぼ比例することとなる．この通信遅延 D_β を最小遅延とする．

Slave Agent が Central Agent と直接通信をしている場合を考えると，ラウンドトリップタイム T_{RTT} は，

$$T_{RTT} = D_\beta^{SC} + D_\beta^{CS} \tag{2}$$

となる．それぞれ右上の添字は S が Slave Agent，C が Central Agent を表し，SC は Slave Agent から Central Agent までの通信路についてであることを示し，CS はその逆を意味する．

Slave Agent が n 個存在し，エージェントはメッセージを受けとったら T_{rest} だけ待って次の通信を開始するとすると，Central Agent のあるホストと Slave Agent のあるホストでは

$$T_{interval}^{C} = \frac{T_{RTT} + T_{rest}}{n} \tag{3}$$

$$T_{interval}^{S} = T_{RTT} + T_{rest} \tag{4}$$

の間隔でメッセージを送り出していることになる．ここで Central Agent と Slave Agent それぞれにおいてメッセージを送信するために必要な処理時間を T_{send} とし，$T_{send}^{CS} > T_{send}^{SC}$ とすると，$T_{interval}^{C} > T_{send}^{CS}$ かつ $T_{interval}^{S} > T_{send}^{SC}$ であるから，n が増大したり T_{rest} が減少したりすることによってこの送信間隔 $T_{interval}$ が狭くなってきたとき待ち時間が発生しないためには，

$$T_{RTT} > nT_{send}^{CS} - T_{rest} \tag{5}$$

$$T_{RTT} > T_{send}^{SC} - T_{rest} \tag{6}$$

である必要がある．ここで，$T_{send}^{CS} > T_{send}^{SC}$ であることより，式 (5) があるので式 (6) は無視してよい．これが満たされない場合は，満たされなかったエージェントにおいて待ち時間 T_{wait} が加えられ，式 (3) は次のように修正される．

$$T_{interval}^{C} = T_{send}^{CS} = \frac{T_{RTT} + T_{rest} + T_{wait}^{C}}{n} \tag{7}$$

このとき，ラウンドトリップタイム T'_{RTT} は，

$$T'_{RTT} = T_{RTT} + T_{wait}^{C} = nT_{send}^{CS} - T_{rest} \tag{8}$$

となる．

次に Slave Agent と Central Agent の間に Dealer Agent を配置してまとめ送りを行った場合について考える．このとき T_{RTT} は，

$$T_{RTT} = D_\beta^{SD} + T_{collect}^{D} + D_\beta^{DC} + D_\beta^{CD} + D_\beta^{DS} \tag{9}$$

となる．ここで $T_{collect}^{D}$ は Dealer Agent でまとめ送りをする際のメッセージの平均集積待ち時間である．

Slave Agent が m 個存在し，メッセージを受けとったら T_{rest} だけ待って次の通信を開始するとすると，各ホストでの平均メッセージ送出間隔は，

$$T_{interval}^{C} = T_{RTT} + T_{rest} \tag{10}$$

$$T_{interval}^{D} = \frac{T_{RTT} + T_{rest}}{1+m} \tag{11}$$

$$T_{interval}^{S} = T_{RTT} + T_{rest} \tag{12}$$

で表される．このシステムの送信待ち時間が発生しない条件は $T_{interval}^{C} > T_{send}^{CD}$，$T_{interval}^{D} > T_{send}^{DC}$，$T_{interval}^{D} > T_{send}^{DS}$，$T_{interval}^{S} > T_{send}^{SD}$ である．ここで $T_{interval}^{C} = T_{interval}^{S} > T_{interval}^{D}$ であり，Dealer Agent は Slave Agent に近いところに配置するので $T_{send}^{DC} > T_{send}^{DS}$ である．したがって，このシステムで待ち時間が発生しない条件は，式 (11) から

$$\frac{T_{RTT} + T_{rest}}{1+m} > T_{send}^{DC} \tag{13}$$

となる．

ここで $m = n$ とすると $T_{send}^{CD} > T_{send}^{DC}$, $T_{send}^{CS} > T_{send}^{DS}$, $T_{send}^{CD} \simeq T_{send}^{CS} - T_{send}^{DS}$ であるから，式 (5), (13) より

$$\begin{aligned} & nT_{send}^{CS} - (m+1)T_{send}^{DC} \\ & > nT_{send}^{CS} - (n+1)T_{send}^{CD} \\ & \simeq (n-1)\left(T_{send}^{CS} - T_{send}^{DS}\right) \\ & > 0 \end{aligned} \tag{14}$$

したがって，式 (5) が成り立たなくても式 (13) が成り立つ状態が存在する．すなわち，Dealer Agent がなくて待ち時間が発生する場合に式 (13) が成り立つ場合に Dealer Agent を配置することで待ち時間が発生しない状態にすることができる．特にすべての Slave Agent を Dealer Agent に引き継ぐのではなく，複数の Dealer Agent に分割すれば，$n > m$ であることから式 (13) の成り立つ範囲が広くなる．

以上より，式 (5) が成り立つ場合に式 (13) が成り立つように Dealer Agent を配置しまとめ送りを開始する．状況が変化し，Dealer Agent がまとめ送りを行っているにもかかわらず式 (5), (13) を満たさない場合には Dealer Agent がいない方がよいので，この Dealer Agent は自分のすべての下位エージェントを上位エージェントに渡して消滅する．

4. 階層的マルチエージェントシステムの実装

本章では前章で説明したシステムの実装例として，実際に研究室において実装した階層的マルチエージェントシステムについて説明する．実装にあたっては IBM Tokyo Research Laboratory で開発され無償配布されている ASDK（Aglets Software Development Kit）ver1.0.3 を使用し，UNIX Workstation 上で動作させた．

図 6 に実装した階層的マルチエージェントシステムのモデルを示す．モバイルエージェントは Aglet クラスを extend した Port クラスと，それぞれの仕事を定義した Agent クラスによって構成されている．中心的な役割を担うエージェントは CentralAgent/Port クラスである（以降 Central Agent と呼ぶ）．Central Agent と通信を行うネットワークに散在するモバイルエージェントは，SlaveAgent/Port クラスのインスタンスとして生成され（以降 Slave Agent），それらを結び付ける Dealer は DealerAgent/Port クラスによって定義されている（以降 Dealer Agent）．

4.1 エージェント管理

3.2 節で述べたように，Dealer Agent は直下の子エージェントのみ管理する．

Dealer Agent は，メッセージ通信の手段を確保するために自分の直接上位の Dealer Agent または Central Agent（以降 Master Agent）の Proxy と，直接下位の Dealer Agent または Slave Agent（以降 Sub Agent）の Proxy のみを保持している．さらに，保持している Proxy の番号とその Proxy が示す Sub Agent の Local Name（後述）を対応させる SubProxyTable を持ち，Sub Agent が移動したり消滅したり生成されたりするたびに自らの Table を更新する．それ以外の Sub Agent のさらに下部や Master Agent より上位でのどんな変化も Table には影響しない（**図 7**）．

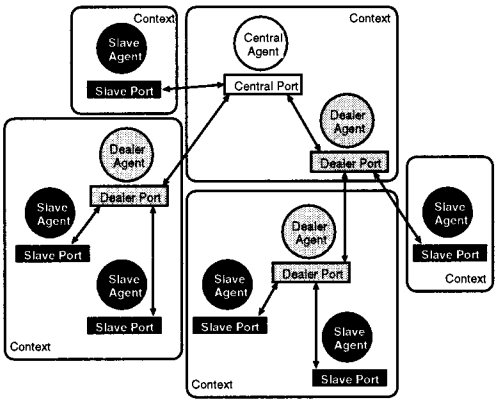


図 6 実装した階層的マルチエージェントシステムのモデル
Fig. 6 The system model of hierarchical multi-agent system.

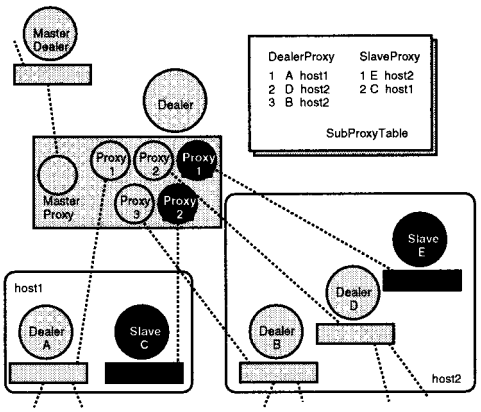


図 7 エージェント管理
Fig. 7 Agent management.

4.2 エージェントの名前付け

エージェントは，新たな Sub Agent を生成する際，および新たな Agent が移動してきた際に，名前が一意になるように名前を割り当てる．このとき割り当てられた名前を Local Name と呼ぶ．システムにグローバルに一意な名前が必要な場合は，一番上位の Dealer Agent から Local Name をつなげて Global Name と

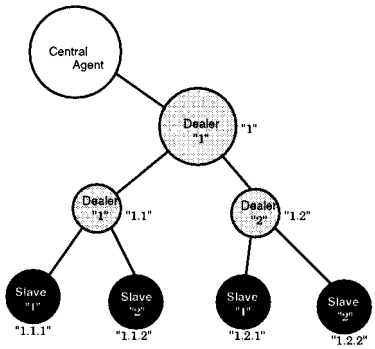


図 8 名前管理
Fig. 8 Agent name management.

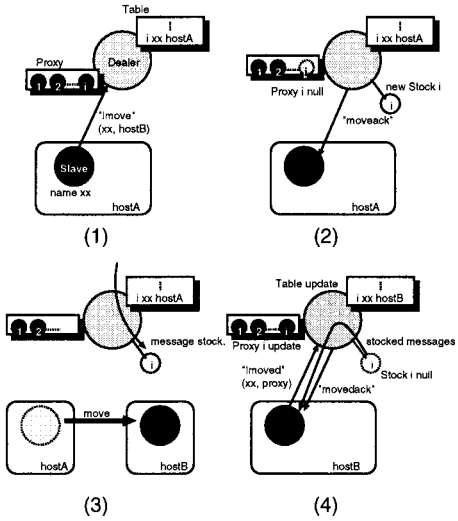


図 9 エージェントの移動
Fig. 9 Movement of agents.

する（**図 8**）.

本実装では，各階層における Local Name は数字で表した．各々の Dealer Agent にとって最初の Sub Agent の名前を “1” とし，以下 “2”，“3”... と続けて名前付けを行う．また Dealer Agent と Slave Agent は区別して扱っているので，Dealer Agent と Slave Agent の名前の重複を許している．

4.3 エージェントの移動機能

エージェントが連続した通信を保ちながら異なるホストに移動する際の手順を**図 9** に示す.

- まず移動の意志を持つエージェント（以降 Move Agent）は，自分に接続するすべてのエージェント（Master Agent とすべての Sub Agent）に対し，その旨通知する (1).
- 移動の意志通知を受けとったエージェント（以降 Connected Agent）は，自分が移動のオペレーションの最中などの都合の悪い場合を除いて，了解の返事を送る．その際，Connected Agent は Move Agent の Proxy を消滅させ，代わりに Move Agent 宛のメッセージを貯蔵する Stock を生成する (2).
- すべての Connected Agent から了解の返事を得た Move Agent は，移動を開始する．この間に Move Agent に宛てて発信されたメッセージは，Connected Agent において Stock に貯められている (3).
- 移動が完了したら，Move Agent はすべての Connected Agent に移動の完了を伝える．移動の完了を受けとった Connected Agent は Move Agent 宛にストックされたメッセージを Move Agent に渡し，Stock を消滅させ，Proxy を更新する (4).

末端の Slave Agent ではなく，Dealer Agent もしくは Central Agent の移動の際も同様にして移動処理を行う.

4.4 階層構造の再構成機能

前節で述べたエージェントの移動を用いて，システムの構造を変化させる際の手順を次に示す．**図 10**, **図 11**, **図 12** は Dealer Agent A が，自分の Sub Agent である Dealer Agent A.x を Dealer Agent A.y の Sub Agent とするように構造を変化させる場合を示している．以下では Dealer Agent A を Sender，Dealer Agent A.x を Mover，Dealer Agent A.y を Receiver と呼ぶことにする．まず，Sender は Mover に新たな Master Agent となる Receiver の Proxy を渡し，Mover は Master Proxy を送られてきた Proxy に更新する．同様に Sender は Receiver に Mover の Proxy を送る．これで Mover と Receiver の間にコネクションが張られたことになる（図 10 右）．また，この際 Sender は Mover の Proxy を消し，代わりに Stock を生成して Mover 宛のメッセージを保持しておく.

コネクションが張られると，Receiver は Mover の新たな名前（ここでは x'）を作成し，Mover に改名を要求する（図 11 左）．この際，自らの SubProxyTable も更新する．このように改名が必要なのは，システムにおいて名前がそのまま構造を表しているためである．Mover に Sub Agent が存在する場合には，Mover は自分のすべての Sub Agent に対し改名を要求する．この改名はその Sub Agent の Master Agent の名前を古いものから新しいものへ置換することで行われる.

改名要求に対し，Sub Agent を持たないエージェントは自分の名前の変化を受け入れた後，Master Agent に対し改名了解の返事を送る．すべての Sub Agent か

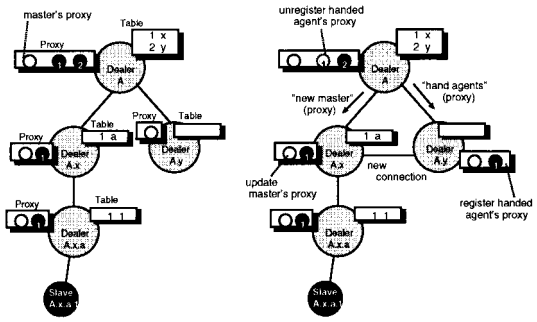


図 10 Sub Agent を Sub Agent に渡す変化 1
Fig. 10 Handover procedure for Agents I.

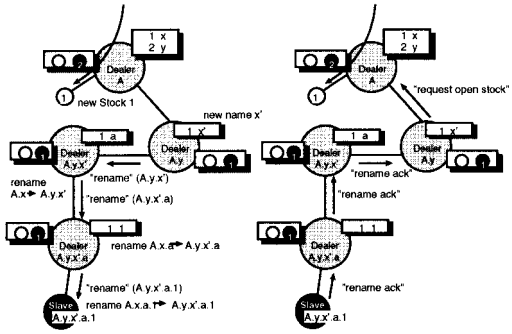


図 11 Sub Agent を Sub Agent に渡す変化 2
Fig. 11 Handover procedure for Agents II.

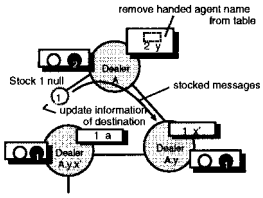


図 12 Sub Agent を Sub Agent に渡す変化 3
Fig. 12 Handover procedure for Agents III.

ら改名の了解を得たエージェントは Master Agent に自らの改名了解の返事を送る．Mover はすべての Sub Agent の改名が完了したら，Receiver に改名了解を通告する．改名了解を受けとった Receiver は，Sender に Mover 向けの Stock に収められたメッセージを渡すように要求する（図 11 右）．

Sender は Stock メッセージを要求されると，Stock の中のメッセージに含まれる目的地情報を変換したのち，Receiver に渡す．すべて渡し終えたら Stock を消滅させ，最後に SubProxyTable から Mover を抹消する（図 12）．

4.5 動的再構成

実装したこの階層的マルチエージェントシステムでは，各 Dealer（Central）Agent は自らが受けとるメッセージをホスト別に集計することで構造を変化させるべきか，そうでないかを判断している．

各 Dealer（Central）Agent は，メッセージを受けとる際，そのメッセージがどのホストから来たかを SubProxyTable の Sub Agent とホスト名の対応表から判断し，前にそのホストからメッセージ群（まとめられているメッセージは 1 とカウント）が来てからの間隔を計算する．今回の実装ではこのメッセージ群受信間隔をモニタしていき，3.3 節の式 (5) が成り立つかどうかをチェックする．メッセージ群受信間隔が狭くなって閾値 $T_{thr} = T^{CA}_{relay} + \alpha$（$\alpha$ はシステムの安定性を保つため必要）より低くなり，式 (5) が偽になると，式 (13) も考慮したうえで Dealer（Central）Agent は SubProxyTable を参照して一番 busy にメッセージを送信してくるホストに新たな Dealer Agent を生成して送り込み，その Dealer Agent にそのホストにいる自分の Sub Agent を渡す．

このときにモニタしているメッセージ群受信間隔は受けとった Message の数を数えているもので，まとめて送られたメッセージは 1 つとしてカウントしているが，Dealer Agent はこれとは別に，まとめて送られたメッセージをまとめられた分だけカウントしている．この正味のメッセージ数によって計算された受信間隔をメッセージ受信間隔と呼ぶ．Sub Agent からあまりメッセージが来なくなって，メッセージ受信間隔が広がって先の閾値 T_{thr} をある程度超えた場合（この閾値を T'_{thr} とする）は，メッセージ受信間隔が一番広い Dealer Agent を消滅させた場合どの程度メッセージ群数が増加するかを計算し，それでもまだメッセージ群受信間隔が T_{thr} を割っているようかどうかを確かめる．もしメッセージ群受信間隔が T_{thr} を割っているならばその Dealer Agent を消滅させ，その Sub Agent を直接自分の Sub Agent にする．

こうしてシステムは動的に再構成される．

5. DealerAgent の効果の実験

はじめにエージェント間におけるメッセージ通信の実際と Dealer Agent によるまとめ送りの効果を確認するため，次のような実験を行った．

5.1 実験方法

図 13 に実験の概略を示す．Slave Agent は Central Agent にメッセージを送り，Central Agent はメッセージを受信したら送信元の Slave Agent に対してメッセージを返信する．Slave Agent は Central Agent からの返信を受けたらある時間だけ待って，次のメッセージを送信する．この待ち時間は負指数分

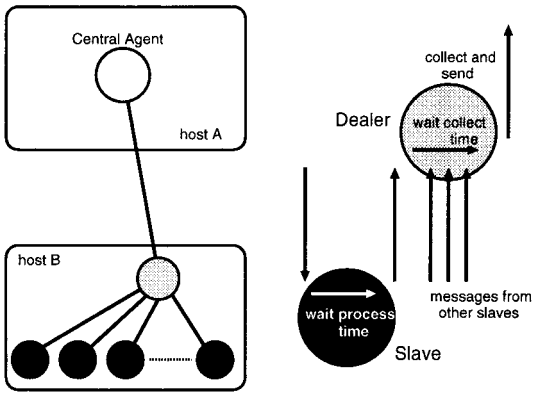


図 13 実験の概略
Fig. 13 An outline of experiments.

布とし，その平均を $T_{process}$ とおく．これは，Slave Agent で何らかの処理を行っていることを模擬している．

Slave Agent が Central Agent にメッセージを送信してから，その返事が戻ってくるまでのラウンドトリップタイムと Central Agent における平均メッセージ（メッセージ群ではない）受信間隔を Slave Agent の数，$T_{process}$，Dealer Agent の有無，また Dealer Agent での集積のための最大待ち時間を変化させて計測した．この最大待ち時間を $T_{collect}$ とおく．Dealer Agent では，最初のメッセージが到着してから $T_{collect}$ が経過するか，自分の下位エージェントの数だけメッセージが到着した（これ以上メッセージが到着しない）ときに Central Agent にまとめられたメッセージ群を送信する．

ここでラウンドトリップタイムを片道の送信にかかる時間ではなく往復の時間として計測したのは，異ホスト間における正確な時間間隔の計測が困難であるからである．

図 13 における host A には Sun Ultra2 model 2296（2cpu）を，host B には Sun Ultra2 model 2200（2cpu）を使用し，深夜に他のプロセスが極力ない時間を選んで実験を行った．host A と host B の処理速度はほぼ同じである．

なお，実装の際のエージェントプラットフォームとしては ASDK（Aglets Software Development Kit）version 1.0.3 [13] を用いた．

5.2 実験結果

前節で説明した実験の結果を**図 14** (a) から**図 15** (f) までに示す．図 14 (a)，図 15 (a) は Dealer Agent を用いず，各 Slave Agent が直接 Central Agent と通信したときを表す．図 14 (b)，図 15 (b) は Dealer Agent は用いているものの，まとめ送りをしなかった場合，それ以降は Dealer Agent での $T_{collect}$ の時間を 10，50，100，500 msec. と変化させた場合についてそれぞれ示している．Slave Agent の数については 1，2，3，5，7，10 の 6 通りについて，Slave Agent における平均 $T_{process}$ は 0，10，50，100，500 msec. の 5 通りについて計測した．

5.3 考　　察

Dealer Agent がいない場合についてまず考察する．図 14 (a) よりメッセージ送信に待ち時間が発生しないときの $T_{RTT} = D_{\beta}^{SC} + D_{\beta}^{CS}$ は約 40 msec. であり，また，図 15 (a) より待ち時間が発生したときの $T_{interval}^{C}$ $(= T_{send}^{CS})$ は約 60 msec. であることが分かる．

ここで最小メッセージ処理時間 T_{send} と D_{β} の関係だが，メッセージの送信にはメッセージ自身には D_{β} の遅延しかもたらさず送信先のエージェントに向けて送信されるが，メッセージを送ったエージェントには T_{send} だけの処理時間がかかるという考察ができる．

図 15 (a) において式 (5) を満たさなくなった状態（メッセージ送信に待ち時間が発生している状態）を $T_{interval}^{C}$ が T_{send} に固定されている部分とすると，$T_{process}$ が 0 msec. の場合は Slave Agent の数が 2 つ以上ですでに該当するのに対し，$T_{process}$ が 500 msec. の場合は Slave Agent が 10 個に増えてもそれほど輻輳していないことが分かる．

輻輳している状態で，図 14 (a) を最小二乗法で近似すると，以下のようになる（以降単位 msec.）．ここで n は Slave Agent の数を表す．

$$
\begin{aligned}
T_{process} &= 0 \quad & T_{RTT} &= 58.9n - 43.9 \\
T_{process} &= 10 \quad & T_{RTT} &= 55.7n - 54.4 \\
T_{process} &= 50 \quad & T_{RTT} &= 56.0n - 109.1 \\
T_{process} &= 100 \quad & T_{RTT} &= 52.1n - 135.8
\end{aligned}
$$

式 (8) から

$$T_{RTT} = T_{send}^{CS} n - T_{rest} \tag{15}$$

であるので，$T_{send}^{CS} = 55$ msec. 程度となり，最初にグラフから読みとった T_{send}^{CS} との誤差は 10%以内である．T_{rest} については

$$T_{rest} = 0.94 \times T_{Process} + 48.1 \tag{16}$$

の関係になっているので，おおよそ 50 msec. のバイアスがかかっていると考察できる．これは式 (3) で考慮しなかった，Slave Agent において $T_{process}$ だけ待ってからメッセージのラウンドトリップタイムを測るために時刻を記録するまでの間隔や，メッセージが返信されてきて到着時刻を記録してから $T_{process}$ だけ待つまでの間隔だと思われる．

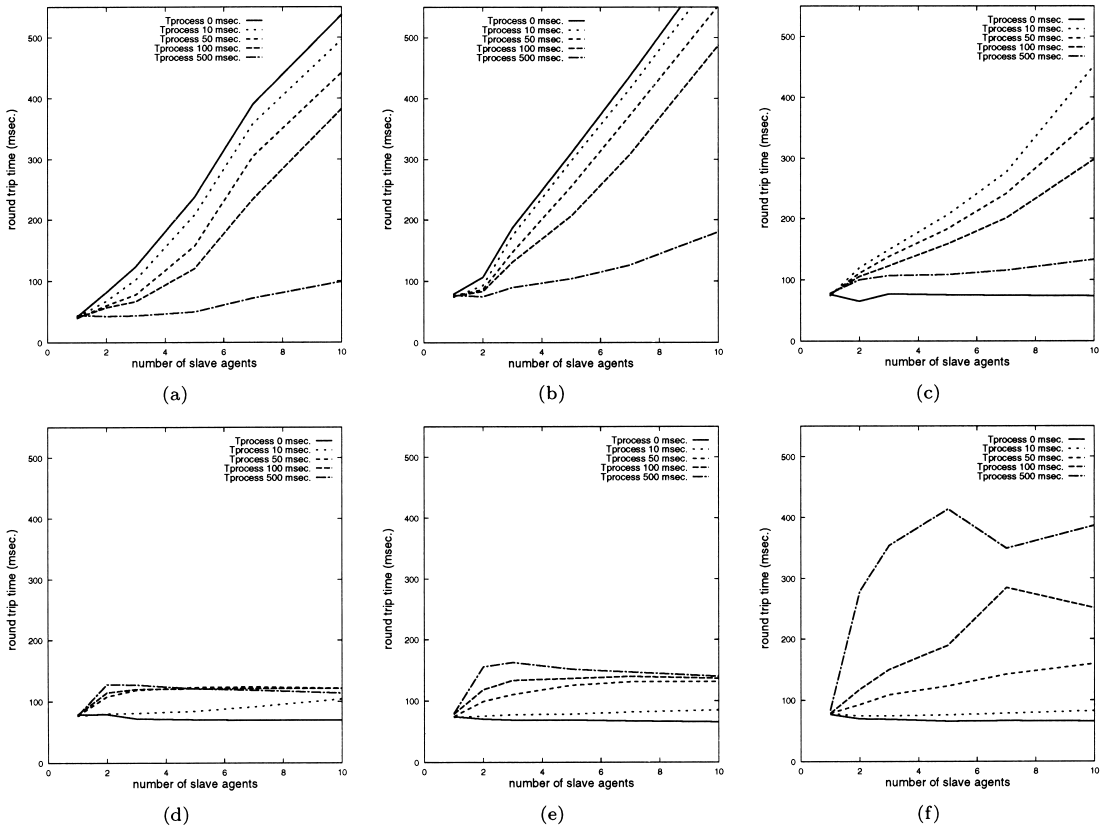


図 14 Slave Agent と Central Agent 間のメッセージの Round Trip Time. (a)：Dealer Agent なし，(b)：Dealer Agent ありかつ蓄積時間=0，(c)，(d)，(e)，(f)：Dealer Agent ありかつ蓄積待ち時間最大 10, 50, 100, 500 msec.

Fig. 14 Round trip time between a Slave Agent and the Central Agent. (a): no Dealer Agent, (b): Dealer Agent without collect time, (c), (d), (e), (f): Dealer Agent with maximum collect time = 10, 50, 100, 500 msec.

以上をふまえて，次は Dealer Agent がいるときの結果を考察する．まとめ送りを行わない場合（図 14 (b)，図 15 (b)）は Dealer Agent—Central Agent 間の通信は Dealer Agent がいない場合と同じなので，図 14 (a) と図 14 (b) の違いは Slave Agent—Dealer Agent 間の通信にかかる処理を表している．図 14 (b) からこの場合の最小遅延 $T_{RTT} = D_\beta^{SD} + D_\beta^{DC} + D_\beta^{CD} + D_\beta^{DS}$ を約 75 msec. と読みとれるので，$D_\beta^{SD} + D_\beta^{DS}$ が約 35 msec. であることが分かる．一方，メッセージ処理時間 T_{send} は図 15 (b) から約 60 msec. で変わらない．これは，通信待ち時間が Dealer Agent では発生せず Central Agent のみで起こっていることを表している．よって，同ホストどうしでのメッセージ処理時間は無視できるほど小さいことが分かる．

輻輳している状態で，図 14 (b) を最小二乗法で近似すると，以下のようになる．ここで n は Slave Agent の数を表す．

$$\begin{array}{ll} T_{process} = 0 & T_{RTT} = 64.0n - 8.2 \\ T_{process} = 10 & T_{RTT} = 61.7n - 12.1 \\ T_{process} = 50 & T_{RTT} = 59.5n - 43.4 \\ T_{process} = 100 & T_{RTT} = 56.5n - 80.5 \end{array} \quad (17)$$

係数はほぼ変わらないので，ここでも同ホストどうしでのメッセージ処理時間は無視できるほど小さいことが分かる．また，

$$T_{rest} = 0.74 \times T_{Process} + 6.5 \quad (18)$$

となるが，先ほど計算した $D_\beta^{SD} + D_\beta^{DS}$ が約 35 msec. あるので，Slave Agent での時刻を記録する際のずれは Dealer Agent がない場合とあまり変わらない．

続いて，まとめ送りを行う場合について考察する．まとめ送りを行う場合は，Dealer Agent における $T_{collect}$ によってやや様相が異なる結果が得られる．$T_{collect}$ が短い場合（図 14 (b)，図 14 (c)）は，集積が

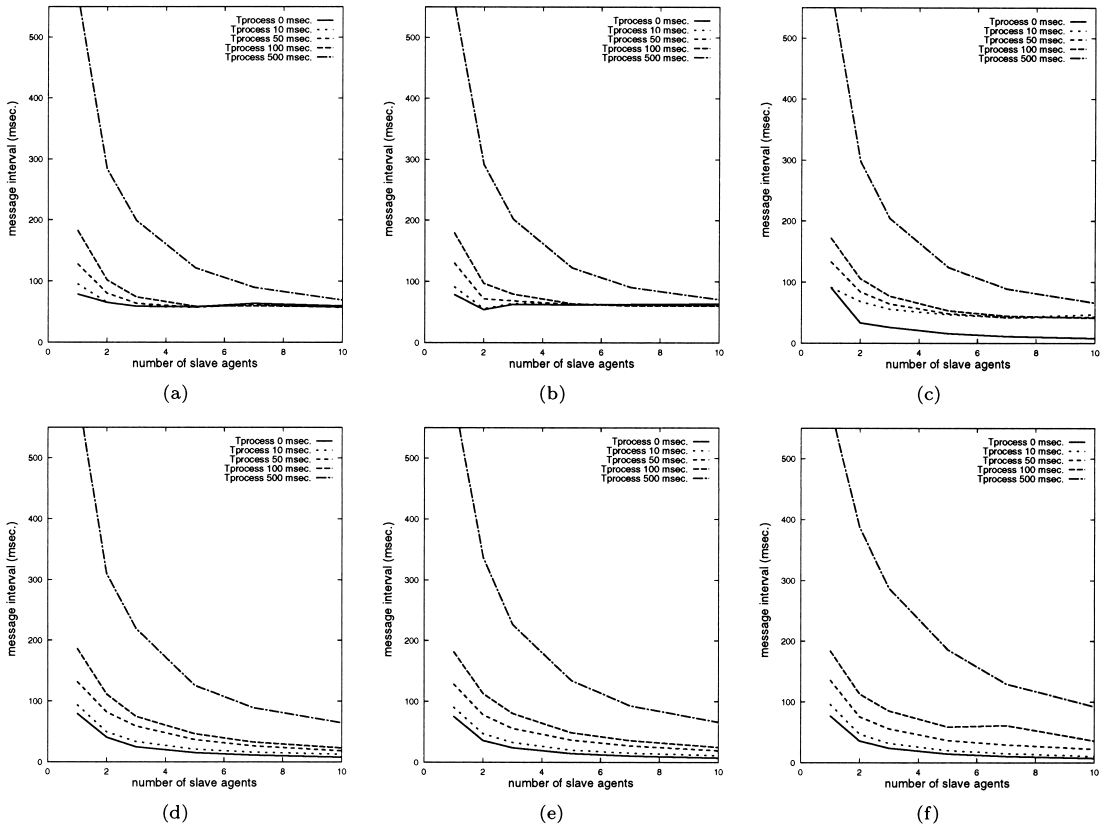


図 15 Central Agent における平均メッセージ受信間隔．(a)：Dealer Agent なし，(b)：Dealer Agent ありかつ蓄積時間=0，(c)，(d)，(e)，(f)：Dealer Agent ありかつ蓄積待ち時間最大 10, 50, 100, 500 msec.

Fig. 15 The average interval for receiving messages on Central Agent. (a): no Dealer Agent, (b): Dealer Agent without collect time, (c), (d), (e), (f): Dealer Agent with maximum collect time = 10, 50, 100, 500 msec.

不十分なためメッセージ数の増加が抑えきれず，輻輳を起こしている．ここで一番トラヒックの多いはずの $T_{process} = 0$ msec. のときに輻輳が起こっていないのは，同じタイミングで Dealer Agent にメッセージが入ってくるためメッセージを集積しやすいからである．図 14 (d)，図 14 (e) ではまとめ送りが効率的に行われて輻輳が生じていないことが確認できる．逆に $T_{collect}$ が長過ぎる場合は（図 14 (f)，図 15 (f)），メッセージ数の増加は抑えられるものの，集積にかかる時間自体がラウンドトリップタイムを悪化させている．

本実験では Dealer Agent のまとめ送りによって Central Agent の輻輳が抑えられたが，Dealer Agent での輻輳も下位に Dealer Agent を生成しまとめ送りすることによって軽減できると期待できる．

6. 動的再構成の実験

続いて，Dealer Agent における動的再構成の様子を確認するため，次のような実験を行った．

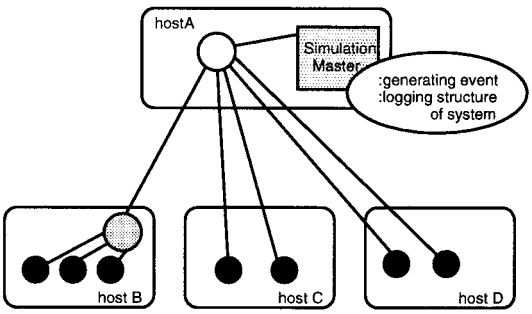


図 16 実験の概略

Fig. 16 An outline of experiments.

6.1 実験方法

図 16 に実験の概略を示す．Central Agent のいるホストに新たに Simulation Master というエージェントを実装し，この Simulation Master がシステムに対し決まったタイミングでイベントを起こしたときにど

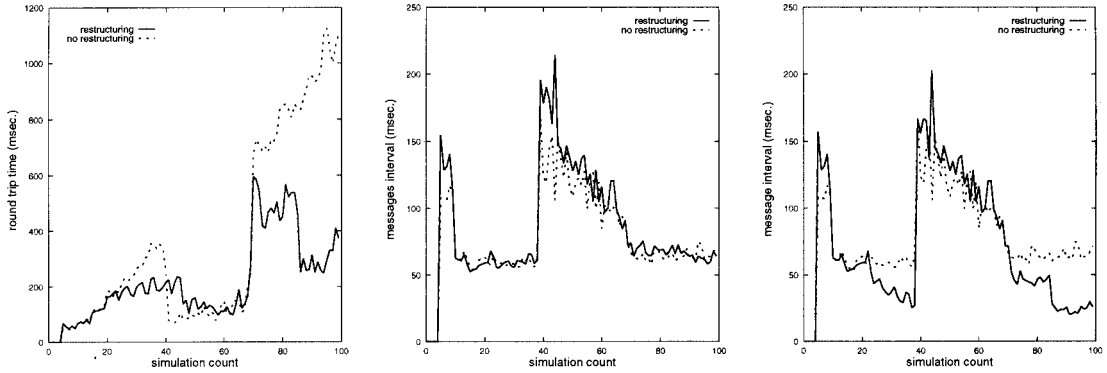


図 17 再構成機能の有無の比較．(a)：ラウンドトリップタイム，(b)：メッセージ群受信間隔，(c)：メッセージ受信間隔

Fig. 17 A comparison of restructuring with no-restructuring: (a): Round Trip Time, (b): Total interval of receiving groups of messages, (c): Total interval of receiving messages.

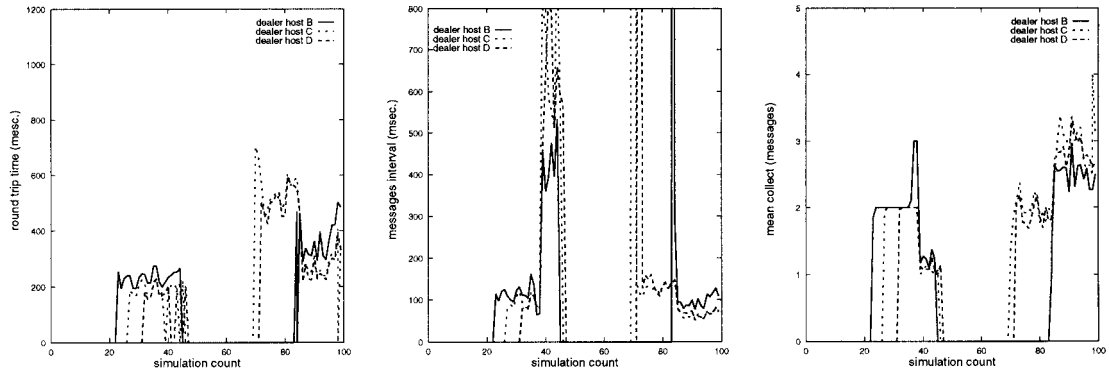


図 18 再構成有りで Dealer Agent のみで集計した (a)：ラウンドトリップタイム，(b)：メッセージ群受信間隔，(c)：平均集積数

Fig. 18 Various measures related to Dealer Agent on each host with restructuring: (a): Round Trip Time, (b): interval of receiving messages, (c): an average number of summarized data.

のようにシステムが動的再構成を行うかを記録した．イベントは 4 秒ごとにカウントされるカウンタによって駆動し，Simulation Master から Central Agent を通じてシステムの各エージェントに通達される．

イベントは，次のように定めた．カウントが進むに従って host B，host C，host D に順番に Slave Agent が生成され，通信を開始する．カウンタが進んである程度イベントをこなすと，今まで 0 msec. だった $T_{process}$ が 1000 msec. に広がる．さらにしばらく経過すると，今度は $T_{process}$ が 100 msec. に縮む．この間，Dealer Agent での $T_{collect}$ は 100 msec. に固定した．

Central Agent の存在する host A は Sun Ultra2 model 2296（2cpu）を用い，Slave Agent，Dealer Agent の配置される，host B，host C，host D にはそれぞれ SPARC Station 20 model 762（2cpu），Ultra1 model 170E（1cpu），Sun Ultra2 model 2200（2cpu）を用いている．閾値 T_{thr} ，T'_{thr} はそれぞれ，host B では 180 msec.，400 mses.，host C では 160 msec.，300 msec.，host D では 75 msec.，150 msec. と定めた．

6.2 実験結果

実験結果を**図 17** (a) から**図 18** (c) までに示す．図 17 (a) から図 17 (c) までは再構成機能があった場合と，再構成なしで Dealer Agent を用いなかった場合において，ラウンドトリップタイム，メッセージ群受

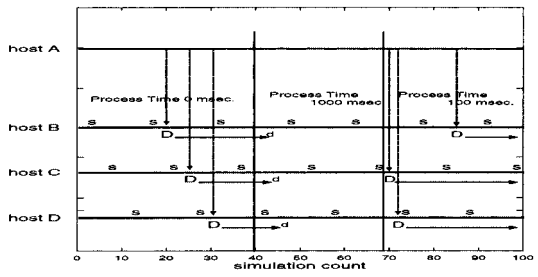


図 19 システムの変化
Fig. 19 transition of the system.

信間隔，メッセージ受信間隔を示している．図 18 (a) から図 18 (c) では再構成機能があった場合において，配置された各 Dealer Agent の様子を示している．それぞれ，各 Dealer Agent のみで集計したラウンドトリップタイム，メッセージ群受信間隔，平均集積数である．Slave Agent から送られてくるメッセージは必ず 1 つずつなのでメッセージ群受信間隔は同一となる．

図 19 はイベントとシステムの変化の概略をまとめたものである．“s” が打たれている時刻にそのホストに新たな Slave Agent が生成されたことを示し，“D” の箇所で再構成が行われてそのホストに Dealer Agent が送り込まれている．この際，Central Agent に所属していてそのホストにいる Slave Agent はすべてこの新しい Dealer Agent に手渡される．

また，simulation count が 40 および 69 のところですべての Slave Agent の $T_{process}$ が切り替わっていることを表す．これにともなって Dealer Agent が消滅し，自分の Sub Agent をすべて Central Agent に渡した時刻を “d” で表している．

6.3 考　　察

図 17 (a) より，システムが状況の変化に即して再構成を行った結果，ラウンドトリップタイムが向上していることが分かる．この実験の間，システムは次のように動作したと考えられる．

count 20 あたりから Dealer Agent が配置されてまとめ送りの効果が表れるが，count 39 に $T_{process}$ が 0 msec. から 1000 msec. に急変したことで集積率が悪化（図 18 (c)），それによりメッセージ受信間隔が Dealer Agent を用いない場合よりも広がってしまった（図 17 (c)）．Dealer Agent，Central Agent はメッセージ受信間隔の増大を検知すると一番受信間隔の広い Dealer Agent から順次 Slave Agent を Central Agent に手渡し，まとめ送りをやめる．この結果ラウンドトリップタイムはまとめ送りをしない水準まで下がる（count 50〜65 付近）．count 69 に $T_{process}$ が 100 msec. まで下がると，再び再構成が起こる．メッセージ群受信間隔が急激に狭まった（図 17 (b)）ことを検知した Central Agent は順次各ホストに Dealer Agent を配置して，Slave Agent を手渡している．この結果再びまとめ送りの効果が表れて（図 18 (c)）メッセージ受信間隔が低下し（図 17 (c)），それにもかかわらずメッセージ群受信間隔は Dealer Agent を用いない場合よりも広くなっている（図 17 (b)）．こうして Dealer Agent を用いない場合に輻輳を起こしていると見られる時間帯（count 10〜40，70〜）に Dealer Agent を用いることにより輻輳を回避することができ，ラウンドトリップタイムの改善に成功している．

7. 階層的マルチエージェントシステムの応用例

階層的マルチエージェントシステムが有効にはたらくのは次のような場合であると考えられる．

- 実時間大量アクセスを集中制御する必要がある場合
- 末端のエージェントの数が多く，広域に散らばっている場合
- 末端のエージェントが移動したり，増減したりする場合

このため，たとえばチャットシステムなどへの応用が考えられよう．

現在インターネットのホームページ上では多くのチャットシステムが運用されている．チャットシステムではユーザのチャットへの入室，退室，発言，読み出しのたびにサーバへのアクセスが発生する．大勢のユーザが参加している際には，ネットワークが混むのと同時にサーバへの負荷の集中のためにレスポンスが非常に低下する．このような場面では，階層的マルチエージェントシステムが活用できると考えられる．

ユーザはフロントエンドとしてのユーザエージェントを持ち，ユーザエージェントが Slave Agent を通してチャットシステムとのやりとりを行い，チャットサーバは Central Agent のところに置く．チャットへの入室，退室はネットワーク各所に分散配置した Dealer Agent が代行し，複数のユーザの入室，退室要求をまとめ送りすることにより，チャットサーバではまとめて入室・退室処理を行うことができる．また，発言，読み出しについても Dealer Agent で中継処理することでトラヒック，負荷の軽減が期待できる．具体的には，読み出しの際は現在のチャットシステムにおいてはユーザとサーバがすべてユニキャストで行っているのに対して，この階層的マルチエージェントシステムを用いると Dealer Agent でデータを複製することに

よりマルチキャストでデータを伝送することが可能となる．さらに，発言時も Dealer Agent のところで複数のユーザの発言をまとめることによって，トラヒックの軽減が期待できる．しかしながら，このまとめ送りの際に蓄積時間を長くしすぎると逆にレスポンスの低下を招くことになる．もう 1 つ，チャットシステムでは時間帯によって参加するユーザの数やユーザの集中するネットワーク上の場所が大きく異なる．したがって，このような動的再構成の機能を持った階層的マルチエージェントシステムは，動的に負荷が集中するところに Dealer Agent を配置することで動的に負荷を軽減させることが期待できる．

8. むすび

本研究ではモバイルエージェントの応用例の 1 つとしてチャットシステムのような大量の実時間アクセスシステムへの応用を検討した．大量のモバイルエージェントをネットワーク内で協調させるシステムを考える場合，すべてに自律的な機能を持たせて完全に分散化させたシステムを考えるよりも，ところどころに機能集中を持たせた方が設計しやすいと考えられる．その意味で本研究で示した階層化マルチエージェントシステムは適度な機能分散によって，サーバへの負荷，通信量を分散させるとともに，分散したモバイルマルチエージェントの管理に向いている．

今後の方針としては，より実際のネットワークの状況に即した条件で実験してみることが必要である．また，実際にマルチエージェントシステムを用いて，大量の実時間アクセスシステムを構築していくことを考えている．

参考文献

1) Vitec, J. and Eds, C.T.: Mobile Object Systems: Towards the Programmable Internet, *2nd International Workshop, MOS '96* (1996). http://www.icsi.berkeley.edu/~tschudin/lncs-1222.html.
2) Pitoura, E. and Bhargava, B.: Building Information Systems for Mobile Environments, *3rd International Conference on Information and Knowledge Management* (1994).
3) Maes, P.: Agents that reduce work and information overload, *Comm. ACM*, Vol.37, No.7, pp.30–40 (1994).
4) Horizon Systems Laboratory: Mobile Agents White Paper. http://www.hri.com/HSL/Projects/Concordia/MobileAgentsWhitePaper.html.
5) Bond, A.H. and Gasser, L.: *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann (1988).
6) 阿部倫之，目黒雄峰，中沢 実，服部進実：自律分散エージェントによるマルチサーバクライアントモデルの動的負荷制御.
7) 小野貴久，荻原 淳，秋吉政徳：リサイクル調整支援へのマルチエージェント技術の適用，マルチメディア通信と分散処理，Vol.86, No.20, pp.115–120 (1998).
8) 荻野長生：マルチエージェント型帯域割り当て方式，信学技報，Vol.IN96-122, OFS96-60, pp.99–60 (1997).
9) 西田豊明：マルチエージェント型知識ベースシステム，*Computer Today*, No.53, pp.6–12 (1993).
10) 桑原和宏，石田 亨，大里延康：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，Vol.J79-B-I, No.5, pp.346–354 (1996).
11) 丸山剛一，酒井和男，渡部智樹，岸田克己：移動エージェントを用いたデータ集約システム—Artemis，第 56 回情報処理学会全国大会（平成 10 年前期）論文集 (1998).
12) 吉川典史，國頭吾郎，相澤清晴，羽鳥光俊：マルチエージェントシステムによる実時間大量アクセスシステム，信学技報 MVE98-92，電子情報通信学会 (1999).
13) IBM 東京基礎研究所：IBM Aglets Software Development Kit – Home Page. http://www.trl.ibm.co.jp/aglets/index-j.html.

(平成 11 年 5 月 11 日受付)
(平成 11 年 12 月 2 日採録)

國頭 吾郎

昭和 47 年生．平成 7 年東京大学工学部電子工学科卒業．現在，同大学大学院工学系研究科電子情報工学専攻博士課程在学中．協調エージェント間通信の研究に従事．

吉川 典史

昭和 49 年生．平成 11 年東京大学大学院工学系研究科電子情報工学専攻修士課程修了．マルチモバイルエージェントシステムの研究に従事．同年ソニー（株）入社．

相澤　清晴（正会員）

昭和 58 年東京大学工学部電子工学科卒業．昭和 63 年同大学大学院博士課程修了．工学博士．現在同大学電子情報工学科助教授．平成 2 年から 2 年間米国イリノイ大学客員助教授．昭和 61 年度丹羽記念賞，電子情報通信学会学篠原記念学術奨励賞受賞．平成元年度同会論文賞および小林ファウンダーズメダル，平成 3 年度業績賞受賞．画像の符号化と処理，コンピュテーショナルセンサ等の研究に従事．IEEE，電子情報通信学会，映像メディア学会各会員．