博士論文

協調エージェント技術に関する研究

指導教官　相澤 清晴 助教授

東京大学大学院工学系研究科 電子情報工学専攻

氏名　77113　國頭 吾郎

提出日　平成 11 年 12 月 17 日

目 次

図目次

表目次

Chapter 1

序論

本論文のキーワードは「モバイルエージェント」である。多様なエージェントの中で本論文ではモバイルエージェントに着目した。本章ではまず、マルチメディアサービスにおけるエージェントの位置づけを明確にし、本論文の構成を説明する。

1.1 本論文の背景

近年の情報技術の発展・普及には目を見張るものがある。移動体通信分野では5年前には高価で珍しかった携帯電話が、今や多くの人が持っていて珍しくなどないものとなった。また移動体通信のキーワードであった「いつでもどこでもだれとでも」は達成されつつあり、今や音声通信ばかりではなくデータ通信も携帯端末の大きなトラヒックとなりつつある。

少し前までは携帯端末を用いたデータ通信は主にショートメールサービス、あるいはキャリアが提供するメールサービスだけで、ごく一部の人たちがノートパソコンやPDAを接続してデータ通信を利用していた。しかしながら、現在はそればかりではなく、NTTドコモグループが提供しているiMode, DDIグループのEZ Web, JPHONEグループのJSkyWebといったサービスを利用し、メールやWWWブラウジングインターネット接続ばかりではなく、銀行のサービス、チケットの予約といった、様々なサービスが携帯端末で利用できるようになってきた。

しかしながら、やはり携帯端末のCPU処理能力には制約がある。そこで、携帯端末上ではなくてもできる仕事を他のCPUパワーを借りて処理することができれば、より高機能のサービスが利用できるようになるであろう。

インターネットの爆発的な流行により、この平成不況の中パソコン市場は好調である。数年前はパソコンは事務機かマニアの持ちものという感じだったものが、今はスタイリッシュでカッコいいものもたくさん登場し、トレンディドラマの中でも単なるインテリアではなくストーリーの中でメールの送受信に使われたりするようになった。大衆に受け入れられたといえるであろう。昔のパソコンと大きく違う点は操作性にあるといえる。「パソコンはソフトを入れればいろいろなことができる。だから使い方が難しい」のは当たり前ではなくなってきているのである。ハード

ウェアも OS もアプリケーションも、ユーザに負担を強いないで使える快適な環境を提供することを競っている。CD を入れれば自動的に演奏を開始することなど当たり前で、ボタン一つ押せば電源が入って自動的にメールを受信してくれるボタンがついている機種まである。

ここで注意する必要があるのは、ユーザが求める簡単な操作性は必ずしも画一的にすることではない、ということである。白物家電もユーザの多様な要望 (興味) に応えるべく、そして付加価値をつけて、より魅力的な商品にするために多機能になりつつある。しかしながら、その代わりに操作性が犠牲となっているものも少なくない。多種多様のユーザの要求、多機能のハードウェア、それらを簡単な操作で使えるようにするといった、ハードウェアとユーザとの仲立ちになるようなものが今後もますます必要となるであろう。

ネットワーク分野ではどんなプロトコルの上でも IP を乗せること、いわゆる IP over Everything で「いつでもどこでも誰とでも IP を」を目指してきた。また、Everything over IP では「何でも IP で送ろう」ということで統合型マルチメディアネットワークを目指してきた。これらにおいてはネットワークは (極端にいえば)IP を送ることが第一であった。しかしながらこれからの Everything over Everything の時代においてはネットワークは単に IP だけではなく、その上のコンテンツやサービスについてもしっかり考慮する必要がある。ネットワークも動的かつ知的になる必要があるのである。

1.2 本論文の目的

一言でエージェント技術といっても、その内容は実に幅広く、多岐に渡っている。本論文ではまずそれらのエージェント技術をいくつかの切り口で整理することを試みている。最も直観的な分類法のひとつは、ユーザインターフェース、協調・交渉、そしてモバイルエージェントの大きく 3 つの分野にわけるものであろう。前節の挙げた携帯端末の高機能化についてはモバイルエージェントが活躍できそうである。ユーザとハードウェアの仲立ちではユーザインターフェース及び協調・交渉が重要になろう。また、知的で動的なネットワークにはエージェントの 3 つの分野が関連することであろう。

本論文は将来の統合マルチメディアサービスにおいてモバイルエージェントが活躍するべく、モバイルエージェントがシームレスに協調するための新たな通信の枠組を提案している。また、情報検索システムへのエージェントの応用を試み、プロトタイプを作成した。これはインターネットでの WEB ブラウジングで重要な役割を担っている情報検索技術をプライベートなシステムに導入し、単にプライベートの検索システムとして利用するばかりでなくエージェントベースで他の検索システムと連携することによってよりグローバルな検索システムになることが期待できるものである。

1.3 本論文の構成

本論文の構成は次の通りである。

第 1 章　序論
第 2 章　エージェント技術
第 3 章　協調エージェント間通信のためのトラッキングエージェントの提案
第 4 章　モバイルエージェント言語とトラッキングエージェントの実装
第 5 章　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
第 6 章　Web とエージェント
第 7 章　モバイルエージェントによる連携検索
第 8 章　結論

図 1.1 は各章の関係を表したものである。本論文の流れを簡単に説明すれば以下のようになる。

第 2 章では、エージェントに関する基礎知識を提供すべく、用途・背景を取り上げ概観している。エージェントという言葉は実に多様な場面に用いられており、これがつかみどころのない印象を与える原因の一つとなっている。本章ではいくつかの分類法を取り上げ研究分野を整理した上で、本論文で着目している「モバイルエージェント」についてどのようなものであるのかを述べる。端的に言えば、モバイルエージェントはいわばネットワーク上を移動しながら仕事を遂行するプログラムと言うことができるが、プログラミングの立場から言えば、オブジェクト指向をさらに進めた新しいパラダイムであると言うこともできる。

第 3 章と第 4 章では、複数のモバイルエージェント間の通信を支援するために提案した新しい枠組みである、トラッキングエージェントについて論じている。モバイルエージェントの応用例としては単独のモバイルエージェントが多くの仕事を行う例が示されることが多い。しかしながら将来のネットワークサービスにおいては、複数のモバイルエージェントが協調しながら仕事を遂行することが予想される。このような場合においてはエージェント間の通信が非常に重要となる。第 3 章ではこのような場合での、効率の良いモバイルエージェント間の通信を実現するために、トラッキングエージェントを用いる新しい通信の枠組みを提案している。続く第 4章[1]ではエージェント言語を用いたトラッキングエージェントの実装例を示している。具体的にはエージェント言語として AgentTcl, Aglets を用いた。

トラッキングエージェントは複数のモバイルエージェントが対等に通信をすることを前提としていた一方で、第 5 章[2]では多対一のエージェント間の通信が主である環境について検討している。本章では階層化エージェントシステムを提案し、多対一通信によって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の実現を示している。さらに階層化構造自体を動的に再構成することにより、システムの状況の変化に柔軟に対応できることも示している。

近年インターネットの急速な普及にともない、目的とするコンテンツを探し出すためのツールである検索システムが重要な役割を担うものとなってきている。広大な Web 空間ばかりではなくユーザのディスクの中のような場所においても、検索システムが必要になるほど扱うコンテン

[1] この研究は奥村恭弘君 (現在 NTT) が卒業研究として中心的に研究を行った。
[2] この研究は吉川典史君 (現在 SONY) が修士論文として中心的に研究を行った。

ツは増大している。第6章、第7章ではこのような検索システムを含むWeb 空間とエージェントの関係に焦点を当てる。これまでは検索システムを構築するためには、強力なマシンパワーと大きなディスク、それから規模の大きなツールを使う必要があった。ところが最近手軽に使える中小規模向けの検索システムが登場し、ユーザを増やしている。本論文では、検索システムとしては大規模なものではなくユーザが手軽にパーソナル用途で使うような小規模なものに着目している。第 6 章では、検索システムの要素技術を論じ、効率的な検索を行うための手法や、プロキシのデータベースの利用などについても論じている。続いて第 7 章では小規模な複数の検索システムがモバイルエージェントを用いて連携することによって、より個々のユーザに応じたグローバルな検索の実現可能性を述べている。

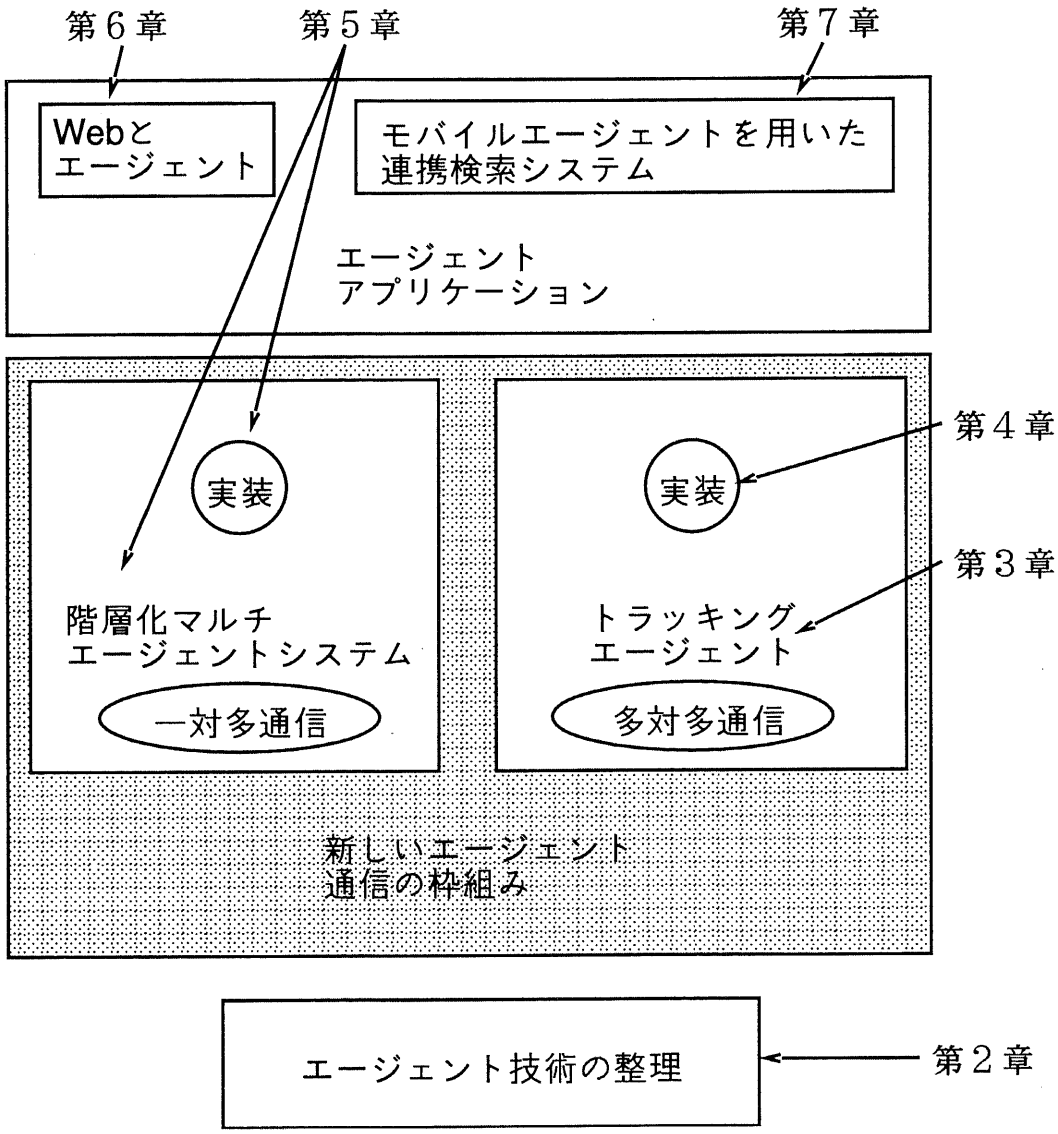


Figure 1.1: 本論文の構成

Chapter 2

エージェント技術

分散人工知能分野において広く用いられてきたエージェントという言葉が近年データベースやソフトウェア工学、通信分野においても使われるようになってきた。本章では、種々の意味を持つようになった「エージェント」研究を概観し広くエージェント像を捉えてみることにする。

2.1 エージェントの現状

様々な領域で使われている、この「エージェント」と言うキーワードが指すものは実に幅広いがそれらはある方向性を持つように思われる。つまり、人間の物理的な活動、あるいは情報活動を支援していく、というものである。言葉を変えれば物理的、情報的シームレス化の支援ということもできよう。

ここでは「エージェント」の現状をいろいろな切り口 で整理する。

2.1.1 エージェントの用法

エージェントを用法の面から捉えてみることにする。エージェントを概念レベル、応用レベル、実装レベルの用法に整理すると、次のようになる。

- 概念レベルの用法

  1. 自律知能を目指す用法
     - 各自の意志決定原理機構に基づき動作する自律的なもの
     - 心的状態を持ち、問題解決や学習機能を持つもの
  2. 分散知能を目指す用法
     - 知性を実現する機能単位
       人間の脳と心の相互作用のような働きである。
     - 協調・交渉などの相互作用を生じさせる基本単位

- 応用レベルの用法
  応用レベルの用法としては次のように 3 つに分類することが出来る。

  1. Softoware Agent
     ネットワーク上に存在する自律的なソフトウェアを指す。ネットワーク上でユーザの代理として働く電子秘書のようなもの、あるいは ネットワーク内でのプログラム間の仲介をおこなうネットワークエージェントのようなものがある。
  2. Interface Agent
     計算機の新たなインターフェースで、人間と計算機間の仲介を行うものをエージェントと呼ぶ。そのユーザの好みを反映することもある。また、擬人化されて表情をもつインターフェースもある。
  3. Believable Agent
     ソフトウェア的な情感を生成し、普通のソフトウェアを超越するような存在感があるもの。擬人化エージェントに感情モデルを組み込んで、エージェントの演技、舞台・背景の制御、カメラワークなどを用いて実在感を高めてユーザを引き込んでしまうことを目的とする。コンピュータとの対話自体を楽しいものにする必要のある、アミューズメントや教育の場面で用いられる。

- 実装レベルの用法
  実装レベルの用法としてはプログラミング言語的な側面から用いられることもある。

  1. Agent Oriented
     エージェント指向のプログラミング。オブジェクト思考の延長上として考えられている。
  2. Mobile Agent
     ネットワーク上を自由に移動することができる。ユーザに必要な情報をネットワーク上から収集したり、ユーザの代理としてネットワーク上を移動して仕事をする。

2.1.2 エージェントの定義

前節のような用法をふまえ、エージェントの定義について考えてみる。

まず「自律的なシステム」の定義を概念レベルで考えると、内面的には「過度の複雑化を避けるために、知識、信念、効用、指向性などの心的用語を用いて振舞いを記述した方が容易な対象」のように定義できる。したがってエージェントは「意志決定原理・機構に基づき、外部から得られた情報に対して、自己の信念や興味に応じて行動するモジュール」といえるであろう。もちろん、自律性の度合はケースバイケースである。

一方、外面から自律的なシステムを定義すると、「系の外からは、自律的に挙動しているように見える機能単位」である。したがってエージェントは、「外部からみれば自律的な挙動を示し、他との協調、共同によって、新しい機能を実現できるもの」といえる。

応用面からみると「ユーザの代理として動作するプログラム」や、「人間の音声や顔、表情を用いて表現することにより、認知的理解を深めてコミュニケーションを容易にする機能」も重要である。エージェントを単に「自律的なシステム」と定義すると、例えば Telescript や java のようにユーザの代理として動作するように書かれたスクリプトプログラムはエージェントとは呼べないように思われるかも知れないが、代理プログラムとはいえ、機能が高くなれば自律性が不可欠であるので、概念レベルの定義と矛盾する訳ではない。したがってこのようなものもエージェントと呼ぶことができよう。

さて次に実装レベルにおける定義であるが、これは非常に難しい。一般に、エージェント指向とオブジェクト指向が比較されるが、現時点ではっきりとしたエージェント指向言語が存在する訳ではない。しかしながら、エージェント指向はコミュニケーションと密接に関連している。異種の知識とデータベースとの間でのコミュニケーションを実現や、エージェント間の交渉プロトコルをサポートするようなものがオブジェクト指向に無いエージェント指向の特徴であろう。

旧来の自律的なシステムは、自律ロボットのような物理的な自立性を問題にしてきたことに比べ、現在のエージェントは主として情報的自立性を問題としている点に注目すべきである。

2.1.3 エージェントの構成

エージェントを広く「自律的なシステム」と捉えると、エージェントには観測、環境モデル、プランニング、行為の 4 つの機能的要素がある (図 2.1)。まずエージェントは周りの環境を観測し、環境モデルを構築する。そして予測を行いプランニングをする。プランニングの過程で、適宜環境モデルを変更していく。そしてその結果からもっとも良い解を取り出し、行為を行う。

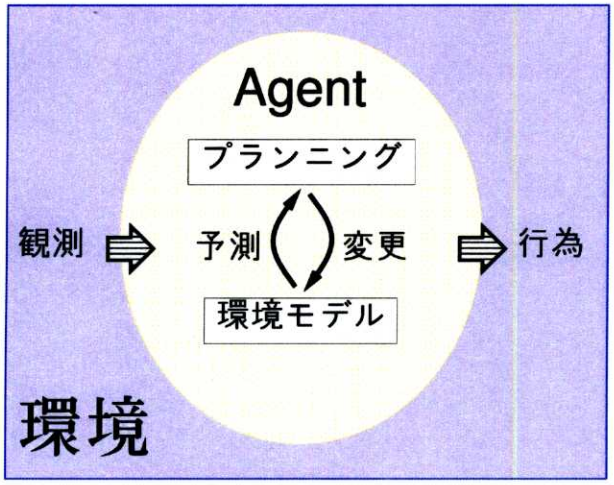


Figure 2.1: エージェントの機能的要素

プランニングの方式には古典的プランニングとリアクティブプランニングの二つがある。

- 古典的プランニング
  環境モデルを変更する規則はオペレータ、すなわち人間に依存する。初期条件から目標条件に向けて、どのような操作をすれば良いのかを考えなければならない。

  この場合の問題点としては、環境の観測の不完全性がある。

  熟考型のプランニングであると言える。

- リアクティブプランニング
  環境の変化に対応するため、自分の行為によって変化した環境を再び観測し、即応できるようにする方法である。環境モデルを変化させるためにリアクティブループと言うものを用いる (図 2.2)。よって人間が操作せずに次の決定を行うことが出来る。

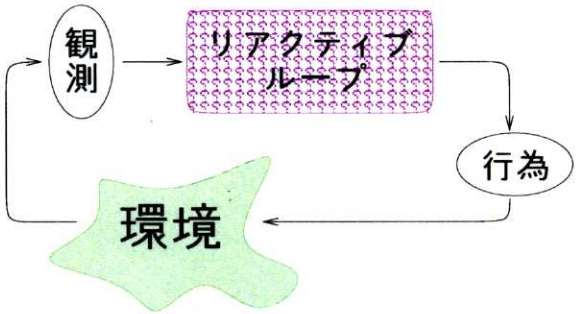


Figure 2.2: リアクティブプランニング

  リアクティブプランニングには、不完全性にどのように対処するか、即応と熟考のトレードオフをどのようにするか、そして再プランニングをどのようにするかが課題となる。

行為に対して報酬を定義し、自動的に再プランニングを行うための方法として「強化学習」という方法がある。次のような順序で学習を行う。

1. 環境を状況認識する
2. 認識した状況から次の行為を選択する
3. 行為を行う
4. 行為に対して報酬を与え、選択した行動、選択の過程から学習し、状況認識に反映させる

教師がいない場合は思考錯誤となる。一般に報酬が与えられるのが遅れることと、観測の不完全性が問題となる。

報酬が最大となる行為を求めることを環境同定型、報酬をもらい続けることを求めることを経験強化型という。

2.1.4 エージェント 3 分野

ここまでいろいろな切り口でエージェントを分類してきた。これらをふまえた上で、エージェントの分野は次のように 3 つの分野に分類すると直感的に理解しやすいであろう (図 2.3)。もちろん完全に全てが分離しているわけではなくお互いに重なっている部分がありうる。

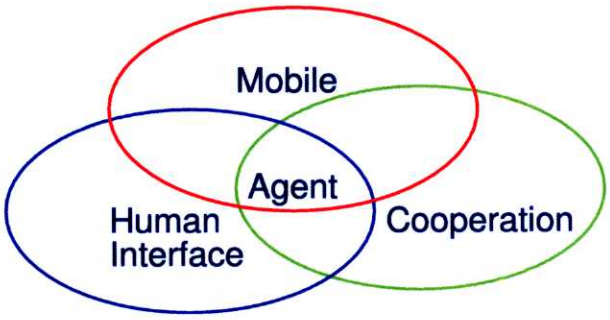


Figure 2.3: エージェント 3 分野

1. 擬人的な化身を用いたヒューマンインターフェース
   外面と内面の両方に人間的なキャラクタを備えたエージェント指向インターフェース [8] である。外面的な擬人性によって、人間がコンピュータとの対話時に感じる心理的な抵抗を軽減し、コンピュータに親近感を持たせることができる。

   エージェントは自己のアイデンティティを表す「顔」や「体」を持ち、音声・表情・身振りのような人間同士の会話で使われる複数のモダリティによって人間とのインタラクションを行う。 これが外面的な擬人性である。応用例としては、ハンバーガー屋の店員の模擬する TOSBURGII[13]、図書館の CG 秘書、顔の表情を中心としたアプローチ [15] などがある。モダリティを豊富に活用することによって、情報のより細かい属性、例えば緊急度、重要度、フォーカスなどを直接的に指示することにも適していると考えられる。

   研究レベルにおいては、外面的擬人性ばかりではなく、内面的な擬人性を持つエージェントを作る試みが始まっている。内面的な擬人性とは次のようなものをいう。

   - 自己概念と持続的な記憶を持つ
   - 自己と他者を区別し、外界を認知することによって自律的に行動する
   - あいづちや、対話の主導権の交替のような人間同士の対話で使われるような習慣をもつ
   - 契約や交渉といった社会的規約、社会通念に沿った行動をする
   - 助けてもらうと感謝するというような、人間の情動的側面をもつ

   内面的擬人性は人間の社会性の一部であるといえる。この社会性を備えたエージェントを社会エージェントと呼ばれている。

2. 交渉・協調・代理人エージェント
   ユーザから与えられた目的と権限に基づいてユーザの行為を代行すると同時に、協調・交渉を行う。ユーザがGUIによってオブジェクトを直接的に操作する直接実行型のインタフェースと対比して、代理人エージェントへのインターフェースは間接実行型と呼ばれることがある。

   ユーザは代理人エージェントに対して、パラメータやキーワード、あるいはスクリプト、あるいは機械学習等によって意図を伝える。スクリプトを作成することは負担であるため、パラメータやキーワードを与えたり、ユーザの振舞いを学習するような機構の方が望ましい。

   応用例としてはユーザの未決メールから重要度の高いものを選別したり、ネットワークからの情報収集においてのフィルタリング、スケジュール管理などがある。

3. モバイルエージェント
   ネットワーク上を移動しながら仕事を行うプログラムである。従来のクライアント・サーバ通信ではクライアントとサーバーにそれぞれクライアントプログラム、サーバプログラムがあって、そのプログラム間のネットワークをデータが往復して通信を行って仕事を進めてきた (図 2.4)。

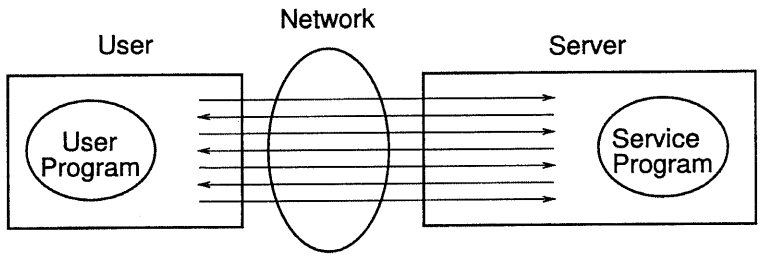


Figure 2.4: 従来のクライアント・サーバ型通信

一方、モバイルエージェントが仕事を行う場合、ネットワーク中をデータが往復する代わりにプログラム自体が移動する。例えばサーバプログラムがクライアント側に移動し、クライアントマシンの中でサーバプログラムとクライアントプログラムが通信をしながら仕事を進めていくというものである。このように、エージェントがネットワーク上を移動し、目的の場所に着くと目的の相手のエージェントとメッセージを交換しあって仕事を行う形態をエージェント型通信と呼ぶ 2.5。

人間がモバイルエージェントを使うという点から考えると、人間が移動する代わりにエージェントがネットワークの中を動き回って仕事をしてくれるということもできる。

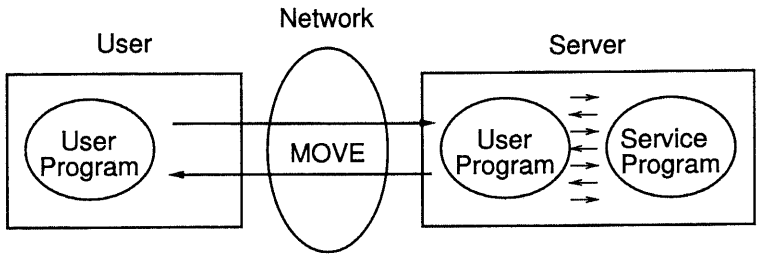


Figure 2.5: エージェント型通信

2.2 エージェントの実際

前節までエージェントの分野の整理を行ってきた。そこで本節ではエージェントを利用した応用例として検討されているものをいくつか取り上げる。

2.2.1 統合メディア通信のためのパーソナルエージェント

近年携帯電話や、PDA 等、多くの携帯情報端末が登場し、一人のユーザが複数の端末を使用することも珍しくなくなっている。パーソナル通信においては端末を意識させないシームレスな通信が求められる [7]。

この研究分野においては「利用者の位置、端末装置の機能の管理、及び個々のユーザに応じたサービスのカスタマイズ」を実現するためにパーソナルエージェント [9]、ユーザエージェント [10] と呼ばれるものについての検討が行われてきた。

ここではユーザが複数の端末を保有する状況で、それらを管理するパーソナルエージェントを想定し、パーソナルエージェントからユーザへの指示が、端末をより意識させないようにするための検討を行った。

2.2.1.1 エージェントモデル

パーソナルエージェントは端末の種類、状況、それらのアクセスルール、個人情報を管理している [12]。また、ユーザインターフェースも提供する。

データとしては音声、テキスト、画像を統合して扱うが、伝送する場合においては区別して扱う。また、ユーザとのインターフェースにおいては、各端末に合った方法でデータを再生、取り込みを行う。

2.2.1.2 統合メディア通信のデータモデル

ここではメディアを次のように 音声データ (S)、テキストデータ (T)、静止画像、図データ (I) の 3 つに分類して考えることとする。端末として表 2.1ようなものを考え、それぞれの端末が扱

うことができるメディアを表にしたものが表 2.1である。

Table 2.1: 端末が扱えるメディア: PDC:携帯電話, PDA:PDA with PDC, Note:Notebook 型パソコン, Desk:Desktop 型パソコン

	INPUT			OUTPUT		
	S	T	I	S	T	I
電話	○	△	×	○	△	×
PDC	○	△	×	○	△	×
PDA	△	○	○	△	○	○
FAX	×	△	○	×	○	○
Pager	×	×	×	×	○	×
Desk	△	○	○	△	○	○
Note	△	○	○	△	○	○

表 2.1をもとに、相手と接続した場合について選択されるべきメディアは表 2.2のようになる。

Table 2.2: 接続メディア

相手	電話	PDC	PDA	FAX	Pager	Note	Desk
電話	S	S	S	T	T	S	S
PDC	S	S	S	T	T	S	S
PDA	S	S	T&I	T&I	T	T&I	T&I
FAX	×	×	I	I	×	I	I
Pager	×	×	×	×	×	×	×
Desk	S&T	S&T	T&I	T&I	T	ALL	ALL
Note	S&T	S&T	T&I	T&I	T	ALL	ALL

2.2.1.3 概念及び動作

User A が User B に向けて発呼すると、あらかじめカスタマイズされたパーソナルエージェントどうしで交渉し、A と B との適切な接続メディアを選択する。複数の接続方法が可能の場合には候補群をユーザに提示する。ユーザはその方式でよければそのまま接続を行う。そして端末に応じた方法でデータを再生する。

発呼したユーザにとっては、発呼するために用いた端末を用いてそのまま通信できた方が望ましい場合が多い。パーソナルエージェントは複数の端末を管理することができるのではあるが、現在の端末の状態やユーザインターフェースを考慮した上で、ユーザへデータの入力方法を指示

する必要がある。

2.2.1.4 具体例

簡単な具体例を示す。

A さんが外出先から携帯電話を用いて B さんに電話をしたが、B さんは会議中であり、B の PA はメールのみ受け取るように指示されていた。

A さんの用事はそれほど急ぎの用事ではなかったが、用件を残したかった。A さんのパーソナルエージェント (PA) は携帯電話でポケベルにメッセージをおくるような操作を指示し、作成されたテキストデータを B の PA へ送った。

B の PA はそのテキストデータをメールの形にフォーマットして、ユーザの端末へ送った。

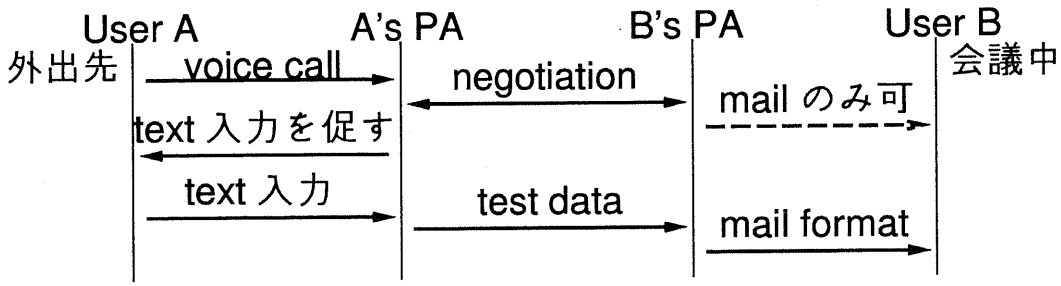


Figure 2.6: 具体例

2.2.1.5 まとめ

このようにユーザが多種の端末やメディアを統合的利用する場合に、ユーザ間での調整をエージェントが状況に応じて行うことによりユーザの負担を軽減しシームレスな通信を実現することができる。

2.2.2 Migrating Personal Agent

本節では、もう一つ別のパーソナルエージェントの例を示す。

将来のパーソナル通信では「いつでもどこでも誰とでも」通信ができることを目指している。この場合、ただ通信できるだけではなく、そのユーザにとってそのときもっとも好ましい環境・条件で通信できることが望ましい。この場合端末が一種類であれば良いが、いろいろなパーソナル通信サービスを一つの端末で実現することは現状では難しく、今後とも複数の端末を使うことになるであろう。したがってある人に連絡を取りたい時に、その人がどんな端末を持っていて、今連絡をとるにはどの手段をとれば良いのか考えなくては、あるいはいろいろ試してみなくてはいけない。

前に述べたようにパーソナルエージェントが個人の環境、サービス条件などを管理しているものである。携帯電話と PDA とを持っている人がいたとしよう。時間帯や場所によっては、例えば会議中には携帯電話ではなく PDA の方にメッセージを欲しいと思う場合もあるだろう。車を運転している時には、緊急の用事はむしろ携帯電話の方に欲しいと思うであろう。

よって、パーソナルエージェントがユーザの使える端末の情報や好ましい通信条件を知っていれば、パーソナルエージェント同士の negotiation によって好みの条件で通信をはじめることができる。

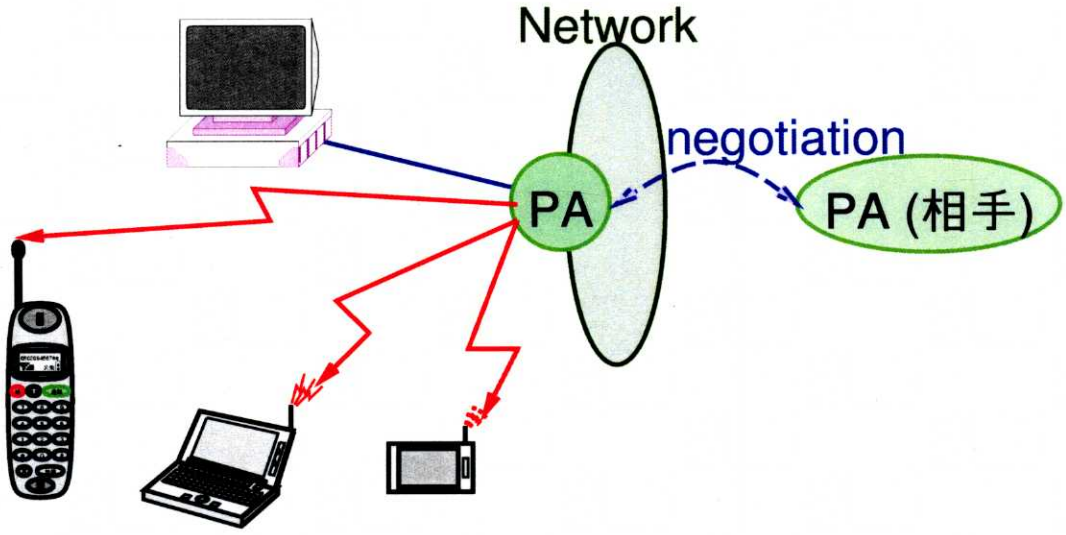


Figure 2.7: PA によるシームレス化

「モーバイルエージェント」は人間が空間を動く代わりにネットワーク内を動き回って仕事をしてくるというものであった。しかし、やはり結局人間も動かなくてはならない。モーバイルエージェントに仕事をさせるために人間がじっとしていなくてはいけないのでは意味が無い。

したがって、人間は動き回るものである、ということを前提にして、シームレス化をはかる必要がある。

ユーザの移動に伴って異種のネットワーク間を自由に移動することができれば水平方向のシームレス化を実現することができるし、またこのパーソナルエージェントに対して多種の端末からアクセスすることができれば垂直方向のシームレス化をも実現することができる。

ここでネットワーク内を自由に移動するパーソナルエージェントがどのようにしてシームレス化を実現できるかについて検討を行う。

2.2.2.1 PA のネットワーク上のモデル

ユーザのデータの全体は Home Memory というところに蓄えられているとする。パーソナルエージェント (以下では PA と略す) はこのデータのうち一部分をコピーして持って動き回る。ここで PA は表 2.3のようなものを管理するものとする。

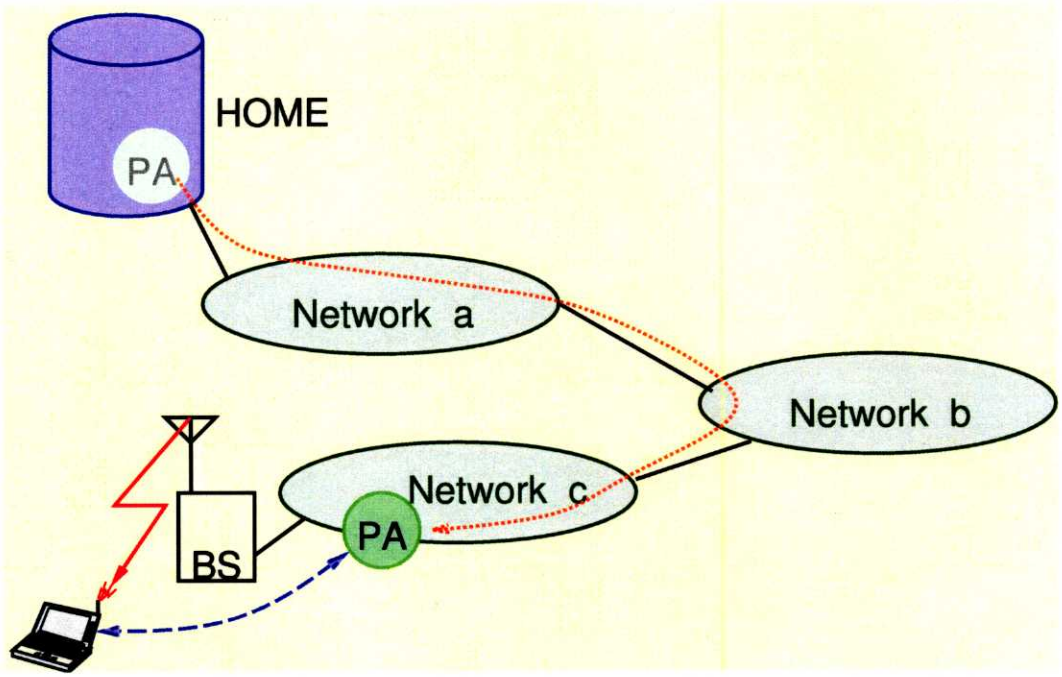


Figure 2.8: Personal Agent Model

Table 2.3: PA が管理する個人情報の例

個人情報	説明
パーソナル ID	個人の識別
登録端末リスト	ユーザが使える端末のリスト
メールボックス	Voice Mail, Text Mail etc 統合して扱う
パーソナルデータ	カレンダーやスケジュール帳、個人の手帳に当たるようなもの
HOME memory の位置	自分の全部のデータのあるところ
暗号鍵	PA の秘密鍵、ユーザその他の公開鍵

PA はユーザがいる近くのネットワークに移動してユーザの個人情報を管理する。したがって

次のような利点がある。

1. ユーザが自分の個人情報に素早くアクセスできる
   自分の手帳は手元にあってすぐに見ることができるから便利なのである。だから手帳の代わりをするようなパーソナルデータはすぐにアクセスできなくてはいけない。

   ネットワーク的に近いところに PA があれば遅延が少ない状態でパーソナルデータにアクセスすることができる。

2. ネットワークにトラブルが起きても自分のパーソナルデータへのアクセスはできる
   ネットワークが混雑している時、あるいは繋がらない時も少なくともパーソナルデータはアクセスしたい。ただし、この場合は HOME との整合性に特に注意する必要がある。

3. ユーザの状態を PA が把握しやすい
   多くの種類の端末を持っているユーザの場合、いまどの端末が使えるか、という情報は重要である。例えば携帯電話が圏内に入っているのか、あるいはユーザがデスクトップ端末のそばにいるのかどうか、といった情報である。

   このような情報はわざわざ HOME からアクセスして調べるよりはユーザに近い位置にいる PA が調べた方が良い。

4. 端末毎のインターフェースを実現
   テキストメッセージが来た場合に携帯には Voice Message として、ポケベルには 文字列で、PDA にはテキストで、端末なら エディタを立ち上げさせる、といったような処理をしやすい。

2.2.2.2 課題

PA が移動する場合について解決しなくてはいけない課題とその解決法を述べる。

- HOME とのデータの整合性
  HOME のデータの一部のコピーを PA が管理するわけであるから PA と HOME のデータが食い違っていはいけない。

  HOME のデータの書き込み権限は PA のみが持ち、PA から HOME へはデータの変更がある場合にのみデータを送るようにする。

- ユーザ、端末の状態の獲得方法
  ユーザ、端末の状態のもっとも確実な獲得方法は、ユーザ自身によるものである。

  携帯端末の圏内情報を元に現在の使える端末の中から着信優先順位をタッチパネルで入力する方法などが考えられる。

- 個人の情報のセキュリティ
  エージェントはパブリックなネットワークの中を動くためセキュリティは大きな問題である。

  PA と自分の端末の間という短い区間であっても、認証をきちんとする必要がある。また、無線の場合には特にデータの漏洩に注意しなくてはいけない。

  秘密鍵による暗号化は鍵を転送する方法が問題となるため公開暗号鍵によって安全なデータのアクセスを実現する。例えばユーザが今日の予定を PA から読み出す順序を図 2.9 に示した。

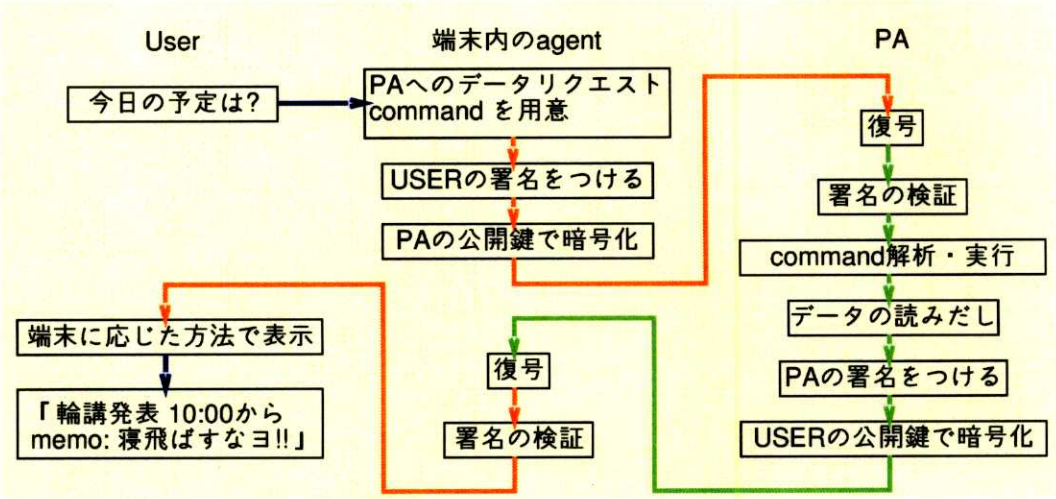


Figure 2.9: 公開暗号鍵を用いたデータの転送例

PA、端末ともメッセージを送る時には、まず自分の署名を入れてから相手側の公開鍵で暗号化して転送する。メッセージを受け取った方はまず復号した後に署名が正しいかどうかを検証する。こうすることで、偽のデータでデータが書き換えられたりすることが無くなり、また伝送路での漏洩も防げる。

- PA がどのようにして移動するか
  PA は小プロセスである。まず自分と全く同じ子プロセスを起動し、それを移動先に転送する。移動先ではそのプログラムが起動すると移動元へ「移動が終了した」というメッセージを送る。親プロセスの方はそのメッセージを受けると exit することで、移動が完了する。

2.2.2.3 モーバイルエージェントとパーソナルエージェント

モーバイルエージェントはユーザの代わりに仕事を行うことが主で、パーソナルエージェントはユーザインターフェースやシームレスなコミュニケーションを提供しているといえよう。

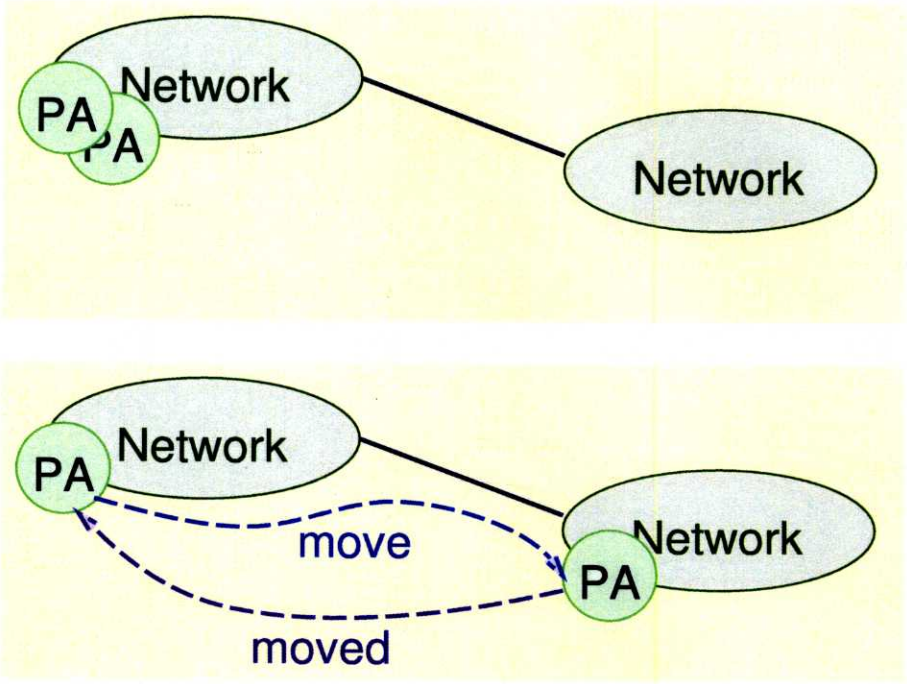


Figure 2.10: Personal Agent Moving

したがって、ユーザまたは端末は直接モーバイルエージェントを送って仕事をさせるのではなく、パーソナルエージェントを介してモーバイルエージェントを使うことでシームレスな通信が実現できると考えられる。

2.2.3 旅行チケット予約システム

モバイルエージェントを用いた応用例として旅行のチケット予約システムがよく挙げられる (図 2.12)。ユーザはあらかじめユーザエージェントに希望する旅行の条件を与えておく。ユーザエージェントはモバイルエージェントをネットワーク上に送り出す。モバイルエージェントはネットワーク上にある旅行会社のサイトを巡り、ユーザが示した条件に合うようなサイトを探し出し、組み合わせるなどアレンジして予約を行う。モバイルエージェントが仕事をしている間、ユーザはオフラインにすることができるので、例えば外出先のようなネットワークが細いところからも利用することが可能である。そして、後でオンラインにしたときにエージェントから予約の結果を受け取ることができる。

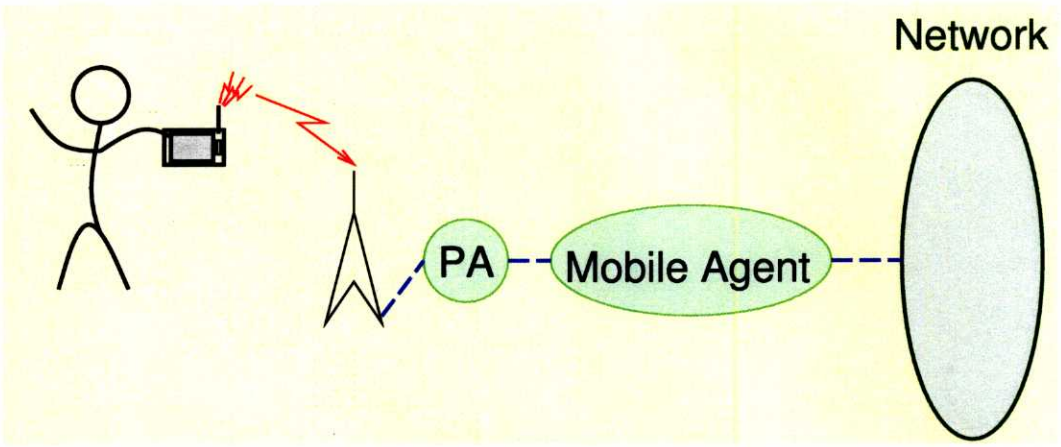


Figure 2.11: パーソナルエージェントとモーバイルエージェント

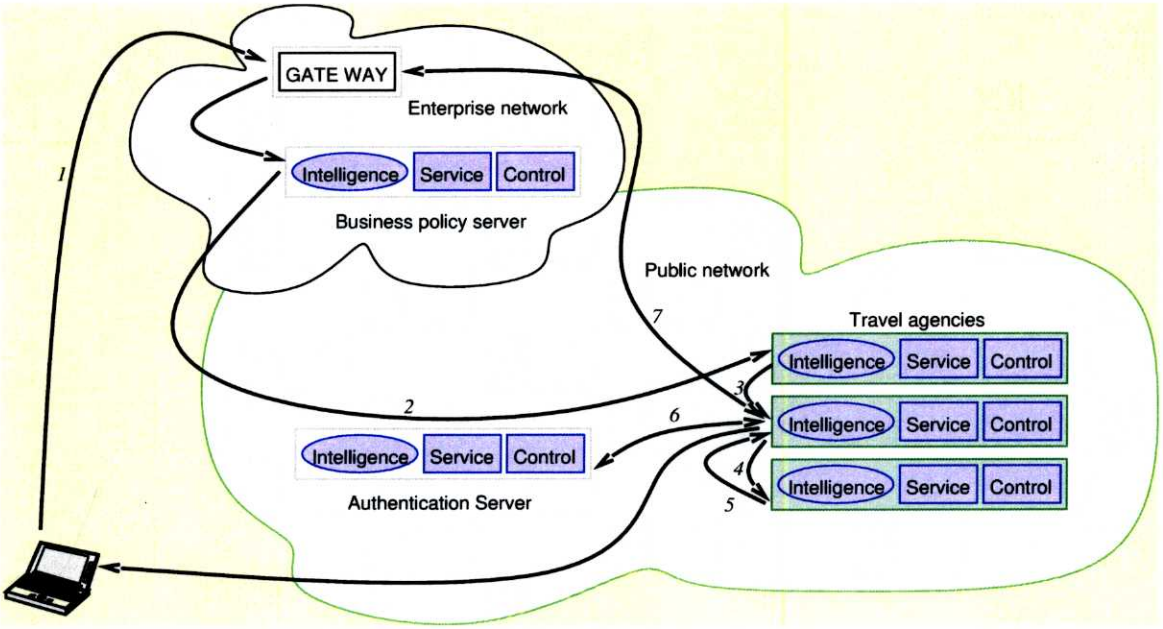


Figure 2.12: Travel reservation scenario

2.2.4 無線系におけるエージェント

これまで述べたようなエージェントはどちらかというとOSIの上位のレイヤでエージェントが働くと考えることができる。一方、無線系の物理層に近いものも検討されていた[11]。

2.2.4.1 移動無線アクセス特有の問題

無線回線を通して移動計算機をインターネットに接続する場合を考えてみる。無線回線では、一般に次のような問題点がある。

1. データ転送速度が低速である
2. 接続が不安定である
3. 通信品質が非常に悪い

1は限られた周波数資源の節約のためである。例えば現行の移動携帯端末の帯域をみてみよう(表2.4)。ようやくPHSで有線のISDNの64Kbpsに追い付いたが、まだLANの速度に比べるとずっと帯域が少ない。

Table 2.4: PDC and PHS

	データ転送速度	音声符号化速度
PDC	最高パケット 64kbps	5.6kbps
PHS	64kbps bps	32kbps

高速の伝送を前提とするインターネットと接続するためには問題であり、WWWをはじめとする多くの画像を転送するアプリケーションを用いる場合に大きなボトルネックとなる。次世代移動通信であるFPLMTSでは2Mbps程度の伝送速度が考えられているが、伝送速度の高速化はなお重要である。

2はシャドウイングやフェージングによって回線が強制切断されたり、発呼しようとしても接続しなかったりする問題である。無線回線を用いる宿命ともいえよう。このような場合にはユーザ側で再発呼・再接続処理が必要であり、またアプリケーションの再起動が伴うこともありユーザには大きな負担となる。これらの処理をユーザに代わって行ってくれるものが求められる。

3も無線特有の問題である。無線の場合はシャドウイングやフェージングによってバースト的にデータに誤りを生ずることがある。伝搬環境は刻刻変化するためこの状況を前もって予測し回避することは困難である。したがって誤り訂正、再送などの処理によって信頼性を高める必要がある。通常誤り制御は下位の無線レイヤで行われることが多いが、誤り制御はスループットの低下を招く。また下位レイヤが誤り制御を行ってくれない場合にはユーザ側が誤り制御を行う必要がある。

ところでインターネット接続を前提とする場合プロトコルの問題が生ずる。インターネット標準の TCP/IP では IP 層では誤り検出しか行わず、再送処理はその上の TCP 層ではじめて行われ、かつ TCP のフレーム長は無線区間のパケット長に比べてずっと長い。したがって、無線レイヤで吸収しきれない誤りは TCP 層に伝搬し、長い TCP パケットの再送を行うこととなり、非常に効率が悪くなる。

インターネット接続をしようとした場合、標準的なプロトコルである TCP/IP に変更を加えることは難しいため、無線レイヤと TCP/IP の間でプロトコル変換を行う必要がある [11]。

なお、PHS データ通信規格 PIAFS では、再送を含むエラー訂正を行うことで IP layer での再送が起らないようになっている。

2.2.4.2 無線アクセスエージェント

無線リンクでインターネット接続をするためには上に述べたように上位レイヤのアプリケーションと下位の無線レイヤの整合を取る必要がある。整合を取るためにパーソナルエージェントの一種である「無線アクセスエージェント」というものを用いた例を紹介する。

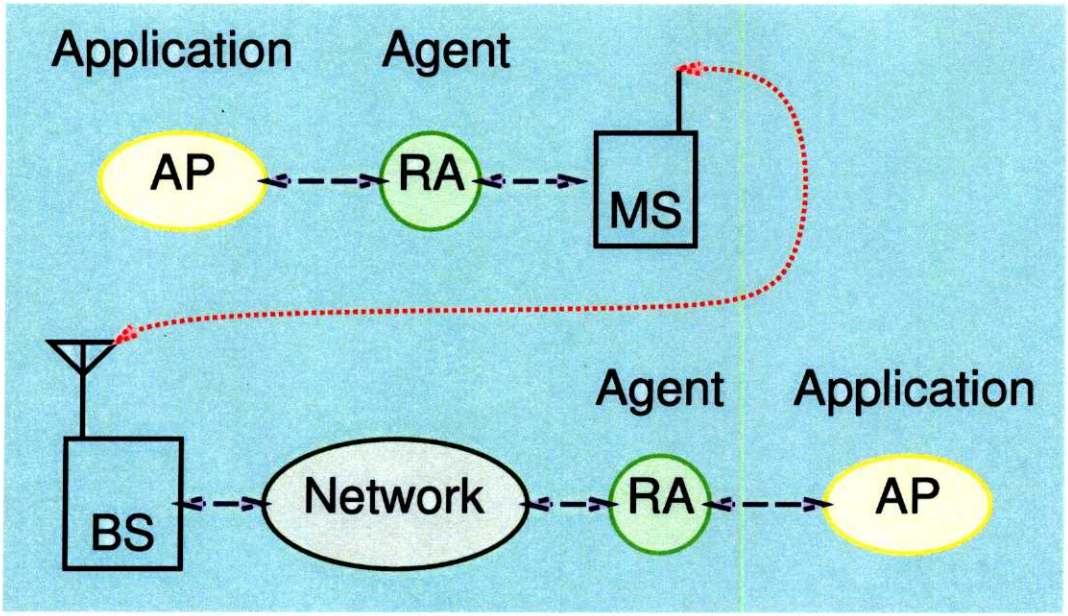


Figure 2.13: 無線アクセスエージェント通信モデル

無線アクセスエージェントは上に述べたようなプロトコル変換を行い、移動端末、固定端末の双方においておく。移動端末においてはアプリケーションからは 無線端末のように見え、無線端末からはアプリケーションのように見える。また固定端末においてはアプリケーションからはネットワークのように見え、ネットワークからはアプリケーションのように見える。

このエージェントは無線固有の問題を解決するために、

- 再発呼、再接続
- 回線状態に応じた伝送制御、動的な誤り制御
- プロトコル変換
- アプリケーションに応じた適応制御

という機能を持つ。

2.3 エージェントの標準化

コンピュータによって人間の活動を支援するエージェント技術は、ネットワークや情報検索における情報フィルタ技術からロボティクス技術まで、実に様々な広い範囲で扱われ始めている。しかし、人間の代理人としての役割という観点からみると多くの共通点が見出せる。これらの多分野の研究者や技術者が集まって、人間とエージェント間、またエージェント同士のインタラクションについての議論を進めることは大変有意義であり、各分野の人達が勝手にインターフェースを定義する前に規格化に向けた検討をしようということで、**FIPA[1]**(Foundation for Intelligent Physical Agents) [1] が設立された。

2.3.1 東京ミーティング

1996 年 10 月に東京でミーティングが開かれた。東京ミーティングでは寄与された様々な種類のエージェントアプリケーションから、エージェント技術の共通項を探るためのターゲットシステムを絞り込んだ。1997 年に FIPA が決めるべきターゲットシステムとしては次の 4 種類のアプリケーションシステムが選ばれた。

1. AUDIO/VISUAL ENTERTAINMENT AND BROADCASTING SYSTEMS (ゲームと放送)
   多チャンネル化する放送分野で制作の生産性向上と、受信側での情報フィルタリングをサポートするシステム。制作側でのエージェントの例としては、企画のための情報収集、ポストプロダクションにおける編集業務や新しい映像効果生成などのサポート等が挙げられる。今回は、受信端末内の番組アーカイブ機能のための情報フィルタリング技術に的を絞った。

2. TRAVEL PLANNING AND MANAGEMENT
   観光案内や旅行のプランニング、飛行機や列車の時刻表の調査と運行状況の把握とチケット購入、ホテルやレンタカーなどの情報提示と予約などの旅行代理店業務を行うシステム。

[1] FIPA の組織:1996 年 9 月 5 日に正式に非営利団体としてスイス・ジュネーブに登記された。議長は MPEG,DAVIC で有名な CSELT(伊) の L.Chiariglione 氏、役員として R.Nicol(BT,UK) 氏、P.Schirling(IBM,USA) 氏、榎並 (NHK) 氏が選出されている。

3. NETWORK PROVISIONING / MANAGEMENT (ネットワーク管理)
   通信ネットワークのモニタリングや料金管理、ホームネットワークの管理、ネットワーク構築時のユーザやオペレータの業務サポートなどを効率良く行うためのシステム。

4. PERSONAL ASSISTANT (電子秘書)
   会合のスケジューリング、メールの優先順位付けや返事・転送、必要なマルチメディア情報の収集や作成・配布等のような秘書的な業務を行うシステム。

この 4 つのシステムに共通するエージェント技術で、1997 年後半で規格化可能な技術を抽出し、これらを実現するための具体的な技術の提案を Call for Proposal として求めることとした[2]。抽出した技術は以下の通りである。

1. エージェント管理関係
   エージェントの名前付け、その存在位置、機能などエージェントインターフェース記述言語や、個人情報をやり取りする場合のセキュリティのあり方、エージェントの生成から消滅までのライフサイクルモデルの設定など。

2. ヒューマン・エージェント間やエージェント間同士のコミュニケーション
   コミュニケーションを実現するためのメッセージの形式、内容記述方式、あるいはプロトコルの設定など。

3. エージェント・ソフトウェア間のインタラクション
   既に存在しているデータベースやサーバーなどの管理ソフトとのやり取りや、映像インデックスされたデータストリームの解釈のための技術。

2.4 まとめ

「エージェント」と呼ばれる分野が非常に広範囲に広がっており、また既存技術との線引きが曖昧である部分も多いために「エージェント」は非常につかみどころのない印象がある。そのせいか、エージェントの存在意義に対して疑問の声が上がることもしばしばである。

本章では多岐にわたるエージェントをいろいろな切り口から整理しながら概観した。大きくユーザインターフェース分野、協調・交渉分野、そしてモバイルエージェント分野に分類すると直観的に整理しやすいと思われる。

筆者は「知的且つ動的なプロセス」という点が既存技術にはない、エージェント独自の特徴であると考え、エージェント 3 分野のうち特にモバイルエージェント分野に着目している。

エージェント技術の普及のためには標準化が求められるが、エージェントの標準化では何を標準化するかが非常に難しいところである。FIPA の今後の活躍に期待したい。

[2] http://drogo.cselt.stet.it/fipa/tokyo/cfp/propos96.htm

Bibliography

[1] Leonardo Chiariglione. Foundation for intelligent physical agents. http://drogo.cselt.stet.it/fipa/.

[2] Inc. General Magic. An introduction to safety and security in telescript, Oct 1995. http://www.genmagic.com/Telescript/TDE/TDEDOCS_HTML/telescript.html/.

[3] Inc. General Magic. The telescript reference reference, Oct 1995. http://www.genmagic.com/Telescript/TDE/TDEDOCS_HTML/telescript.html/.

[4] Robert S. Gray. *Agent Tcl: Alpha Release 1.1.* Department of Computer Science, Dartmouth College, Dec 1995.

[5] H. Lei and D. Duchamp. Transparent file prefetching, 1995. http://www/mcl.cs.columbia.edu/papers/sosp95-submisshion.ps.gz.

[6] Ramachandran Ramjee, Thomas F. La Porta, and Malathi Veeraragahvan. The Use of Network-Based Migrating User Agents for Personal Communication Services. *IEEE Personal Communications*, Vol. 2, No. 6, pp. 62–68, Dec 1995.

[7] 飯田一朗. パーソナル移動通信のためのソフトウェア. 電子情報通信学会誌, 第 78 巻, pp. 150–155. 電子情報通信学会, Feb 1995.

[8] 長尾確. マルチモーダルインタフェースとエージェント. 人工知能学会誌, Vol. 11, No. 1, pp. 32–40, 1996.

[9] 西ヶ谷岳, 飯田一朗. エージェント指向のパーソナル通信網:duet. 信学技報, Vol. CS94-64, pp. 1–6, Aug 1994.

[10] 竹田憲司, 谷英明, 西田竹志. パーソナル通信サービスに向けたユーザエージェントの構成. 信学技報, Vol. CS94-217, pp. 63–68, Mar 1995.

[11] 藤野信次, 竹間智, 飯田一朗. エージェントによる移動無線アクセスの一検討. 信学技報, Vol. RCS95-168, , Feb 1996.

[12] 我妻新吉, 園田雅文. エージェントによる次世代通信システム『メディア統合通信システム』の運用. 信学技報, Vol. CS94-216, pp. 57–62, Mar 1995.

[13] 竹林洋一. 音声自由対話システム tosburg ii—ユーザ中心のマルチモーダルインタフェースの実現に向けて—. 電子情報通信学会論文誌, Vol. J77-D-II, No. 8, pp. 1417–1428, 1994.

[14] 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, Vol. CS95-133, , Nov 1995.

[15] 武部崔. 顔の 3 次元モデルを用いた顔面表情の分析. 電子情報通信学会論文誌, Vol. J74-D-II, No. 6, pp. 766–777, 1991.

Chapter 3

協調エージェント間通信のためのトラッキングエージェントの提案

近年ネットワーク上を移動しながら仕事を行うモーバイルエージェントの注目が高まっている。モーバイルエージェントをネットワーク上で動的に移動させることによってネットワーク上の資源を有効に利用することが可能である。本論文では複数のモーバイルエージェントが協調して一つの作業を進める場合に必要とされるエージェント間の通信のための新しい仕組みを提案する。複数のモーバイルエージェント間の通信を支援するために“トラッキングエージェント”と称するエージェントを用い、これにより 複数のモーバイルエージェントが互いの位置や移動を意識しないシームレスなエージェント間通信の実現が可能である。さらにトラッキングエージェント自身の移動によって、モーバイルエージェントがネットワーク上に偏って存在する場合におけるメッセージの遅延の軽減が期待できる。

3.1 まえがき

インターネットの急速な普及に伴い、ユーザのネットワーク環境は大きく変化している。ネットワーク上には多様なサービスが提供され様々な情報が存在するが、現状は情報の洪水や多くの通信手段の存在による混乱等によってユーザにとって必ずしも使い勝手の良いものとは言えない。様々なユーザの要求に柔軟に対応するには、もはや既存のハードウェアやソフトウェアだけでは困難になってきている。したがって、個々のユーザに応じた、個人とネットワークの仲立ちが必要である。このような秘書のような機能を実現するための一つのアプローチとしてエージェントを用いることが近年研究され、FIPA[1],OMG[3] でエージェントの標準化の作業も始まっている。

エージェントとは現在非常に広い意味で使用されているが、本論文においてはネットワーク上を動的に移動し処理を行うモーバイルエージェントに着目する。モーバイルエージェントに関しては単独のプロセスの動作という観点から論じられることが多い。これに対し、複数のエージェントが協調して一つの作業を進める協調エージェントについても研究が進められつつある [12]。

ネットワーク上に移動・分散しているエージェントを協調作業させる場合、複数の移動するモーバイルエージェント同士の通信をどのようにするかが問題となる。

筆者らは、ネットワーク上を移動する複数のモーバイルエージェント同士が互いの位置や、移動を意識せずに通信するための手法を提案した [9]。トラッキングエージェントはモーバイルエージェントに対し、(1) 協調している他のモーバイルエージェントの位置を意識させない。(2) 協調している他のモーバイルエージェントの移動を意識させない。(3) 必要な時に動的に生成し、動的に消滅する。これにより、シームレスなエージェント間通信の実現が期待される。さらにモーバイルエージェントがネットワーク上に偏って存在する場合には、トラッキングエージェント自身がモーバイルエージェントの近くへ移動することによって、メッセージの遅延を軽減させる事ができる。

本章では 3.2 で提案方式であるトラッキングエージェントについて述べ、3.3 でトラッキングエージェントを介したモーバイルエージェント間の通信方式、およびモーバイルエージェント の移動方式について述べる。3.4ではトラッキングエージェントの移動方法を述べ、3.5ではトラッキングエージェントの有効性を計算機シミュレーションで評価し、3.6でまとめる。

3.2　トラッキングエージェントの提案

3.2.1　モーバイルエージェント

モーバイルエージェントとは次のような能力をもつプロセスとして定義する。(1) ネットワーク上を移動することができる。(2) 移動先で移動前の処理を継続することができる。(3) 離れた他のエージェントと通信を行うことができる。(4) 他のエージェントと通信し協調して仕事をすることができる。(5) 子エージェントを生成することが出来る。(6) 使用できる資源、寿命に制限があり、子エージェントにも同様の制限を設けることができる。

これらの能力を備えていることで、ネットワークを動的に移動して協調作業が可能になる。

通常モーバイルエージェントとはネットワーク上でデータをやり取りする代わりに、プログラム自体を転送することによってクライアントサーバ型通信に対して利点があると考えられている [7]。したがって、モーバイルエージェントの通信相手は固定されたプロセスであると考えられていることが多い。

しかしながら、モーバイルエージェントを用いたシステムの実現に当たっては、モーバイルエージェントの権限、利用可能資源はセキュリティ等の観点から非常に厳しい制約を受ける事が予想される。したがって、単独のモーバイルエージェントが順番に複数の仕事を遂行する事は困難となり、複数のモーバイルエージェントが協調しながら仕事を行わざるを得ない状況が発生すると考えられる。そこで本論文では、モーバイルエージェントが複数で協調作業を行う場合の通信について考察を進める。

3.2.2　協調エージェント間の通信

複数のモーバイルエージェントが協調するためには、相手のモーバイルエージェントに正しくデータが送られなければならない。これらのモーバイルエージェントがネットワーク上を移動する場合、何らかの方法で位置を管理する必要がある。

移動するモーバイルエージェント間で通信を行う場合、(1) エージェント間の直接通信、(2) 中継方式の二つに大別できると考えられる。エージェント間の直接通信としてはいわゆる TCP/IP のブロードキャストのような協調するエージェント全てに移動情報を送る方法が考えられる。しかし、直接通信の場合は協調するモーバイルエージェントの数の増加に伴い、非常に効率が悪くなる。

一方、中継して通信する方式としては次の 2 手法が考えられよう。(1) モーバイルエージェントはそれぞれ HOME と呼ばれるそのモーバイルエージェントの位置を記録するものを持ち、他のモーバイルエージェントは相手先のモーバイルエージェントの HOME に向けてデータを送る。携帯電話の位置登録や Mobile IP[5] などがこの範疇に入る。(2) ネットワーク上に黒板をおき、それぞれのモーバイルエージェントは黒板を読み書きして通信する方式も考えられる [11]。これに加えて筆者らは、第 3 の方式を提案する。その方式は (3) 協調しているモーバイルエージェントだけの位置情報を持ち、メッセージの中継を行うエージェントを必要に応じて動的にネットワーク上に作る方式である。

この中継を行うエージェントは、協調しているモーバイルエージェントだけの位置情報を持ち、必要に応じて動的に生成、消滅するため資源の有効利用が図れる。また、この中継を行うエージェント自身もネットワーク上を移動することができる。

OMG/MASIF の MAF Finder[2] は、Agent system のまとまりを region とし、一つ、もしくは複数の region に対して MAF Finder が存在する。一方、トラッキングエージェントは、協調するモーバイルエージェントのグループ毎に生成する。場所単位の存在ではなく、アプリケーションに応じたタスク単位で生成し、モーバイルエージェントの移動に伴い動的に自身も移動するという点で MAF Finder とは異なる。

以下本論文では、この中継を行うエージェントを “トラッキングエージェント” と呼び、その詳細を以下に記す。

3.3　トラッキングエージェントを介したエージェント間通信

3.3.1　トラッキングエージェント

トラッキングエージェントを用いたエージェント間通信をする場合、エージェント間の関係は図 3.1のようになる。

モーバイルエージェント間の通信はトラッキングエージェントを介して行うものとする。そのためにトラッキングエージェントは各モーバイルエージェントから (1) モーバイルエージェントのいるホストの物理的位置 (2) 協調しているエージェント間で用いる実効的な ID についての情

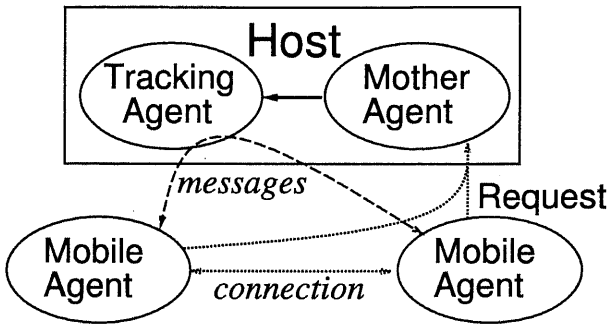


Figure 3.1: エージェント間の関係

報を得る必要がある。

モーバイルエージェントがトラッキングエージェントに登録されると実効 ID が割り当てられ、モーバイルエージェント同士はこの実効 ID を用いて通信を行う。トラッキングエージェントはモーバイルエージェントから受け取ったメッセージを該当するモーバイルエージェントに転送する。

また、モーバイルエージェントがネットワーク上に偏って存在する場合には、トラッキングエージェント自身がモーバイルエージェントの近くへ移動することによって、メッセージの遅延の低減が期待できる。

3.3.2　マザーエージェント

このエージェントはトラッキングエージェントを生成するためのエージェントである。トラッキングエージェントを生成する資源がある各ホスト上に存在し、移動はしない。周辺のある一定の範囲にブロードキャストパケットを出して、マザーエージェントの位置をモーバイルエージェントに知らせることができる。

3.3.3　トラッキングエージェントの生成と消滅

トラッキングエージェントは次のように生成される。まず複数のモーバイルエージェントが協調開始を決定し、トラッキングエージェントを生成するホストを決定する。次にマザーエージェントにトラッキングエージェントの生成要求を出す。マザーエージェントはトラッキングエージェントの生成に必要な情報をモーバイルエージェントから受け取り、自分と同じホスト上にトラッキングエージェントを生成する。

モーバイルエージェントが協調する必要がなくなると、トラッキングエージェントへ登録を削除するように要求する。トラッキングエージェントは登録されているモーバイルエージェントの数が 1 以下になると自動的に消滅する。

3.3.4　モーバイルエージェントの移動

トラッキングエージェントは管理しているモーバイルエージェントの位置情報を把握しなくてはならない。このため、モーバイルエージェントは図3.2のような手順で移動する。まずモーバイルエージェントはトラッキングエージェントに移動することを通知し、トラッキングエージェントからのメッセージを停止し、移動先に子エージェントを生成する（図 3.2a）。移動先の子エージェントが動作すると、トラッキングエージェントに対して子エージェントを生成したことを通知する（図3.2b）。さらにトラッキングエージェントからそのACK が返って来た後、移動元に移動が成功したことを知らせる。その後子エージェントを生成した後に遅れて移動元へ届いたデータを子エージェントに送る（図3.2c）。そして移動元のエージェントは消滅してエージェントの移動が完了する（図 3.2d）。

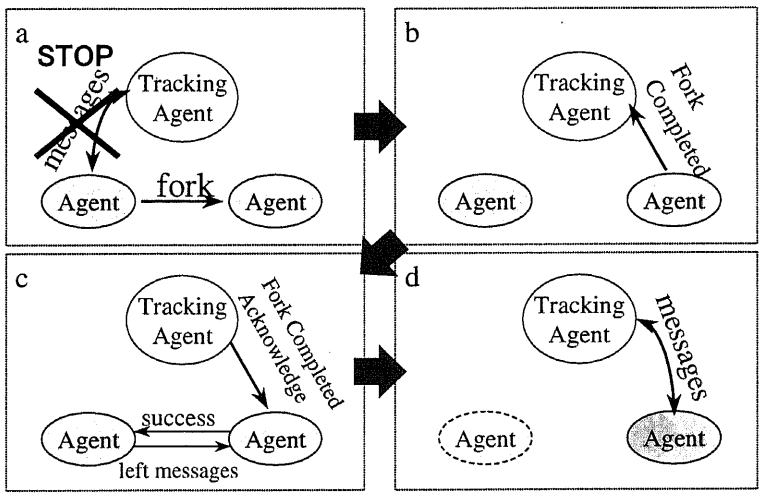


Figure 3.2: モーバイルエージェント の移動手順

3.4　トラッキングエージェントの移動

モーバイルエージェント同様にトラッキングエージェントも移動することができる。モーバイルエージェントが共同作業をする場合、物理的な位置が偏ることも想定される。このような場合には、中継拠点であるトラッキングエージェントもモーバイルエージェントの近くに移動することで、より遅延の少ない通信が可能になると期待される。ここではトラッキングエージェントの移動について検討する。

3.4.1　偏在したエージェント間の協調

複数の協調モーバイルエージェントが偏在する場合、HOME のように固定した中継拠点を用いる方式では、エージェントと HOME とが離れている場合、メッセージは必ず HOME を経由するために経路が長くなる。一方、提案方式では、中継拠点を移動させることでよりモーバイルエージェントに近いところにトラッキングエージェント自身が移動することにより、経路を短くすることができる。したがってモーバイルエージェントが偏在する場合には、移動するトラッキングエージェントを用いると HOME を用いる場合に比べて遅延が小さくなることが期待できる。

3.4.2　トラッキングエージェントの移動アルゴリズム

モーバイルエージェントは図 3.3のような手順で移動する。

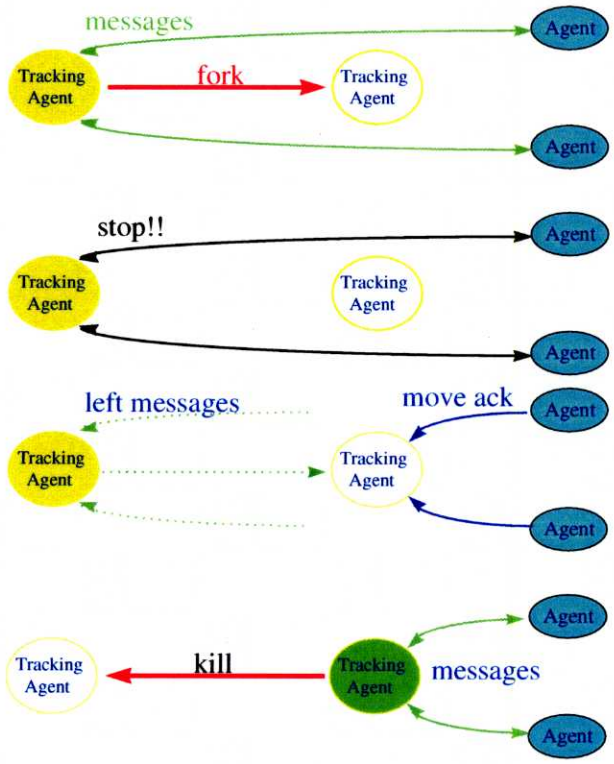


Figure 3.3: トラッキングエージェントの移動手順

まずトラッキングエージェントは中継している全てのモーバイルエージェント移動することを通知し、モーバイルエージェントからのメッセージを停止し、移動先に子エージェントを生成する。移動先の子エージェントが動作すると、モーバイルエージェントに対して子エージェントを生成したことを通知し、モーバイルエージェントからその ACK を受け取る。次に移動元に移動が

成功したことを知らせる。その後子エージェントを生成した後に遅れて移動元へ届いたデータを子エージェントに送る。そして移動元のエージェントは消滅してエージェントの移動が完了する。

3.5 トラッキングエージェントのシミュレーションによる評価

本節では簡単な計算機シミュレーションを用いて、ネットワーク上におけるトラッキングエージェントの効果を評価する。ブロードキャスト方式、HOME を用いる方式、トラッキングエージェントを用いる方式の 3 方式について送信パケット数、送信パケット遅延時間、モーバイルエージェントの移動に要する時間を比較検討する。

3.5.1 モデル

次のようなネットワークのモデルを仮定して検討する。格子状のネットワークを仮定し、格子点（ノード）にホストがあるものとする。それぞれのエージェントは格子点上ホスト上に存在することができ、位置を座標で表すことができる。一つのノードには複数のエージェントが存在することができる。

このネットワーク上の 2 点間の距離は、この格子に沿った道のりで定義する。パケットがこの格子を通る速さは一定とすると、隣の格子点までのパケット到達時間及び移動可能なエージェントが隣の格子点までの移動時間は一定となる。この時間を Δt とする。2 点間の距離そのものがかかる時間を表す事になるこの考え方は IPv6[4] を参考にした。

3.5.2 モーバイルエージェント の移動に要する時間

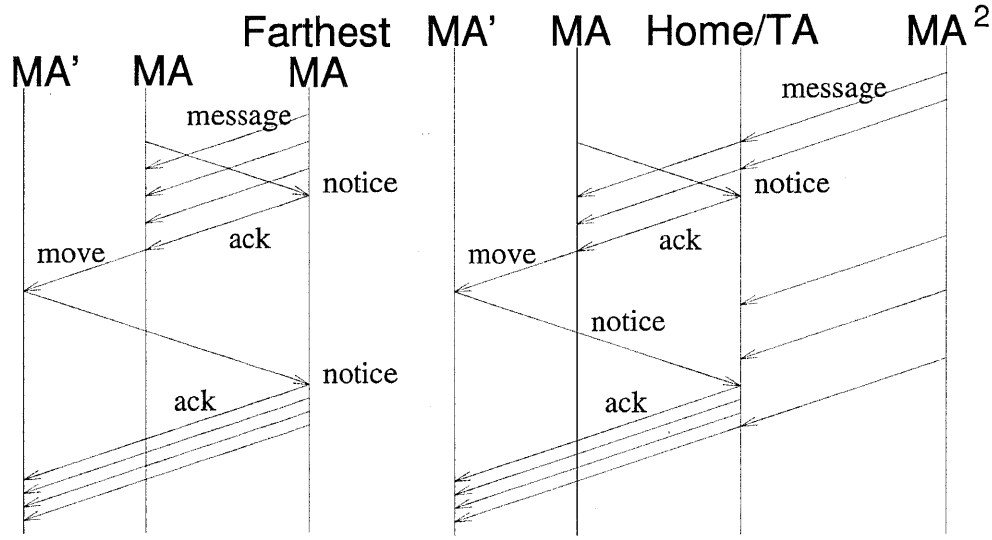
3 方式について モーバイルエージェントの移動に要する時間を評価する。

モーバイルエージェントが移動を決定してから移動終了の通知が到達するまでの時間は次のような式で表すことができる。

$$T_{leave} \times 2 + T_{move} + T_{reach} \tag{3.1}$$

たとえばブロードキャスト方式（図 3.4(a)）を用いる場合モーバイルエージェントは移動することを決定すると全ての協調しているモーバイルエージェントに移動することを通知し、その ACK をもらう。その後移動し、移動したことを通知するとモーバイルエージェント間の通信が再開される。したがって、(3.1) 式において T_{move} は移動時間、T_{leave}, T_{reach} はそれぞれ移動前のモーバイルエージェントから最も遠いモーバイルエージェントまでの移動決定通知到達に要する時間、移動後のモーバイルエージェントから最も遠いモーバイルエージェントまでの移動終了通知到達に時間となる。

次に HOME 及びトラッキングエージェントを用いる方式について検討する（図 3.4(b)）。この 2 方式はいずれも中継してメッセージを転送し、モーバイルエージェント移動は中継点にのみ通知する。HOME 方式では中継点が HOME であり、まずモーバイルエージェントは移動を決定



(a) ブロードキャスト方式　　(b) HOME 方式とトラッキングエージェント方式

Figure 3.4: モーバイルエージェント移動時のパケットの流れ

すると HOME に通知し ACK をもらう。その後移動し、移動したことを通知すると HOME からモーバイルエージェントへのメッセージの転送が再開される。したがって、(3.1) 式において T_{move} はモーバイルエージェントの移動時間、T_{leave}, T_{reach} はそれぞれ移動前のモーバイルエージェントから HOME への移動決定通知時間、移動後のモーバイルエージェントから HOME へのの移動終了通知時間となる。

トラッキングエージェントを用いる方式では、中継点がトラッキングエージェントであり、同様な手順で行われる。

3.5.3　トラッキングエージェントの移動先決定アルゴリズムについて

先のネットワークモデルの下でトラッキングエージェントの移動先の決定を以下のように行う。

トラッキングエージェントの移動先はそれぞれのモーバイルエージェントからデータの伝送にかかる時間の和がなるべく小さくなるように選択する必要がある。トラッキングエージェントはモーバイルエージェントの位置を把握しているため、モーバイルエージェントとトラッキングエージェントとの距離を測ることができる。モーバイルエージェントの数を N 個とし、k 番目のモーバイルエージェントの座標を (x_k, y_k)、トラッキングエージェントの座標を (x_{TA}, y_{TA}) とする。トラッキングエージェントの移動先の候補としてはモーバイルエージェントの重心を選ぶことに

する。トラッキングエージェントの移動先の候補を (x'_{TA}, y'_{TA}) とすると、

$$(x'_{TA}, y'_{TA}) = \frac{1}{N}\sum_{k=1}^{N}(x_k, y_k) \tag{3.2}$$

と表すことができる。

トラッキングエージェントの現在位置から移動先の候補までの距離を d で表すと

$$d = |x'_{TA} - x_{TA}| + |y'_{TA} - y_{TA}| \tag{3.3}$$

となる。

移動前のトラッキングエージェントの位置から全てのモーバイルエージェントまでの距離の和を Sum、移動先の候補から全てのモーバイルエージェントまでの距離の和を Sum' とすると、

$$Sum = \sum_{k=1}^{N} |x_{TA} - x_k| + |y_{TA} - y_k| \tag{3.4}$$

$$Sum' = \sum_{k=1}^{N} |x'_{TA} - x_k| + |y'_{TA} - y_k| \tag{3.5}$$

移動前のトラッキングエージェントから最も遠いモーバイルエージェントまでの距離を d_{MAX}、移動先の候補から最も遠いモーバイルエージェントまでの距離を d'_{MAX} とする。全てのモーバイルエージェントが移動していない状態にあると仮定すると、トラッキングエージェントが移動決定してから移動を開始するまで、すなわち全てのモーバイルエージェントに移動を通知しその ACK を受け取るまでに要する時間は $d_{MAX} \times 2 \times \Delta t$ となる。

したがって、トラッキングエージェントが移動決定から全てのモーバイルエージェントに移動終了通知が到達するまでの時間を D とすると $D = (d + d_{MAX} \times 2 + d'_{MAX}) \times \Delta t$ である。一つのモーバイルエージェントの Δt 時間におけるパケット送信確率を p とすると時間 D にパケットを送信するモーバイルエージェントの数の期待値は $D \times N \times p$ で与えられる。

モーバイルエージェントはトラッキングエージェントの移動決定通知を受け取ってから移動終了通知を受け取るまでの間はトラッキングエージェントにパケットを送ることができないため遅延が発生する。これがトラッキングエージェントの移動によるオーバヘッドとなる時間である。モーバイルエージェントはランダムにパケットを発生させることからこの遅延の大きさの期待値は $D/2$ である。

したがって、トラッキングエージェントの移動に伴って生じるパケット遅延の総和の期待値は

$$\begin{aligned} & D \times N \times p \times D/2 \\ = \ & \frac{(d + 2d_{MAX} + d'_{MAX})^2 Np}{2} \end{aligned} \tag{3.6}$$

である。したがって、$Sum - Sum'$がこの値より大きければトラッキングエージェントが移動するメリットがある。このことから、トラッキングエージェントの移動を決定する条件は

$$\begin{aligned}&\sum_{k=1}^{N} |x_{TA} - x_k| + |y_{TA} - y_k| \\ &- \sum_{k=1}^{N} |x'_{TA} - x_k| + |y'_{TA} - y_k| \\ &> \frac{(d + 2d_{MAX} + d'_{MAX})^2 Np}{2}\end{aligned} \tag{3.7}$$

である。

3.5.4　シミュレーションモデルの詳細

各エージェントが送信するパケットのサイズは全て同じ大きさで、時間Δtあたりに 1 つだけ送信できるものとする。また、移動中にはパケットを送信することはできないものとする。ネットワークの広さは 256 × 256 とし、ネットワークは飽和しないものとする。また、モーバイルエージェントの数は固定とし、移動先を乱数を用いて決定する。HOME 方式での固定中継点の位置は一様乱数を用いて決定する。

モーバイルエージェントのふるまいは以下のパラメータで決定されることとした。

- ネットワーク上のモーバイルエージェント数
- 時間 Δt におけるパケット送信確率
- 時間 Δt における移動確率
- モーバイルエージェントの移動先の決定の乱数（一様分布、非一様分布の 2 種）

モーバイルエージェントの分布に関しては、一様分布の場合と偏在する場合を扱っている。非一様乱数としては正規分布 $N(0.5, 0.25)$ および $N(0.5, 0.125)$ に従う乱数[1]を用いた。なお、ネットワークの大きさを $[0,1] \times [0,1]$ に正規化して扱うこととする。

3.5.5　シミュレーション結果

以下に、計算機シミュレーションの結果を述べる。ここで 3.5.5.1の考察に際してはモーバイルエージェントは一様分布を仮定し、3.5.5.2では分布に偏りのある場合を扱う。

[1] $N(\mu, \sigma^2)$ の密度関数は $f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/(2\sigma^2)}$
ここで用いた乱数は $0 \leq x \leq 1$ とし、乱数 $x < 0$ の場合は $x = 0$、$x > 1$ の場合は $x = 1$ とした。

3.5.5.1　モーバイルエージェントが一様分布する場合

単位時間Δtにおけるモーバイルエージェントの送信確率を変化させた場合のブロードキャスト方式、HOME 方式、トラッキングエージェント方式それぞれのパケット遅延時間のグラフが図 3.5である。時間Δtにおける移動確率は 0.01、モーバイルエージェントの個数は 10 で固定とした。

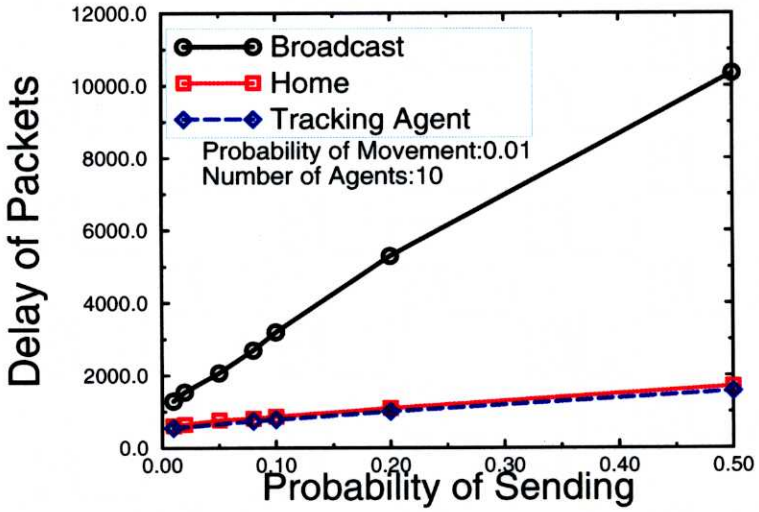


Figure 3.5: パケット送信確率と遅延時間

ブロードキャスト方式は移動時に送信するパケット数も多いため、送信確率が上がると送信キューに溜るパケット数の増加のために遅延が大きくなっていると考えられる。HOME 方式とトラッキングエージェント方式との差はほとんど無い。

次にモーバイルエージェントの数の変化の影響について考察する。図 3.6はモーバイルエージェントの数を変化させた時のパケット遅延のグラフである。時間Δtにおける移動確率は 0.01、パケット送信確率は 0.2 とした。

この場合もやはり図 3.5の場合と同様に、ブロードキャスト方式の遅延の大きさが顕著である。モーバイルエージェントの数が増加すると、送信確率の増加以上にキューに溜るパケットの量が増大し、遅延が大きくなっていると考えられる。

一方、図 3.7はモーバイルエージェントの数の変化による移動に要する時間への影響である。時間Δtにおける移動確率は 0.01、パケット送信確率は 0.2 とした。この場合も同様にブロードキャスト方式は圧倒的に遅延が大きく、モーバイルエージェントの数の変化の影響を大きく受けている。図 3.6、3.7のパケット遅延、移動遅延とも HOME 方式とトラッキングエージェント方式での差はほとんど無い。

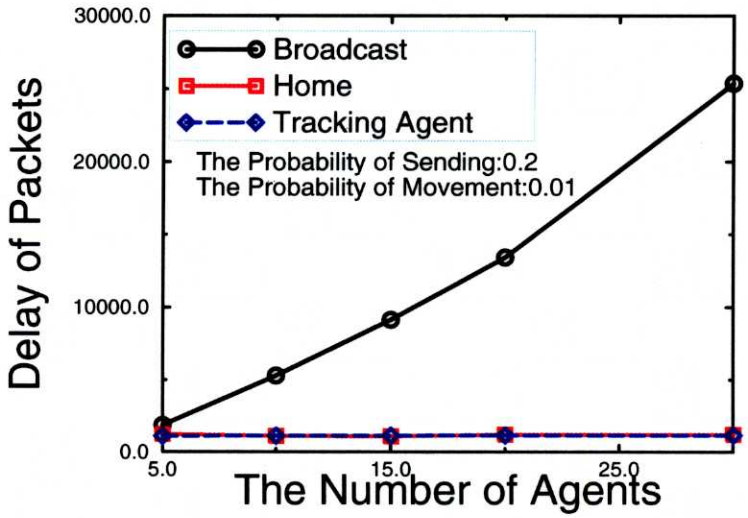


Figure 3.6: モーバイルエージェント数の変化によるパケット遅延の影響

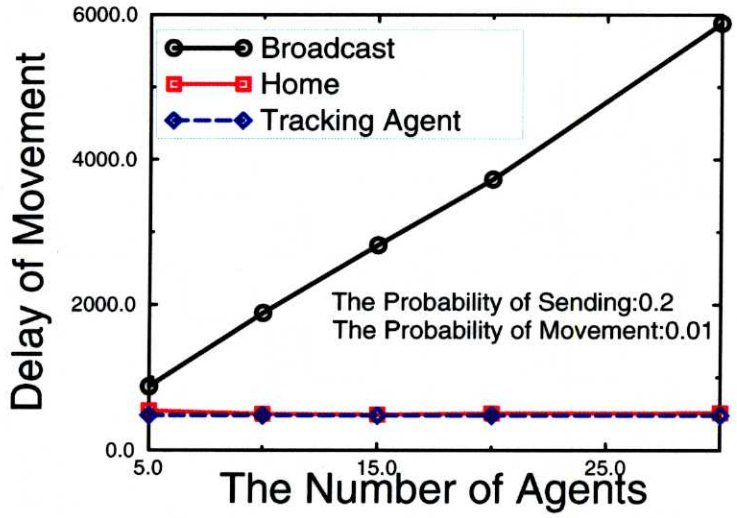


Figure 3.7: モーバイルエージェントの数の変化による移動遅延の影響

3.5.5.2　モーバイルエージェントの分布が偏っている場合

前節までにおいては 3 つの方式を比較したが本節においては HOME を用いる方式とトラッキングエージェントを用いる方式に絞って検討する。

モーバイルエージェントの分布が偏っている場合には、モーバイルエージェントがの偏りが中継点から遠い場合に遅延が大きくなってしまう。このため、トラッキングエージェントがモーバイルエージェントに近付くように移動する事ができれば、遅延を小さくする事ができると考えられる。一方 HOME 方式では中継点が固定のため、偏り位置に応じて遅延が大きくなる。

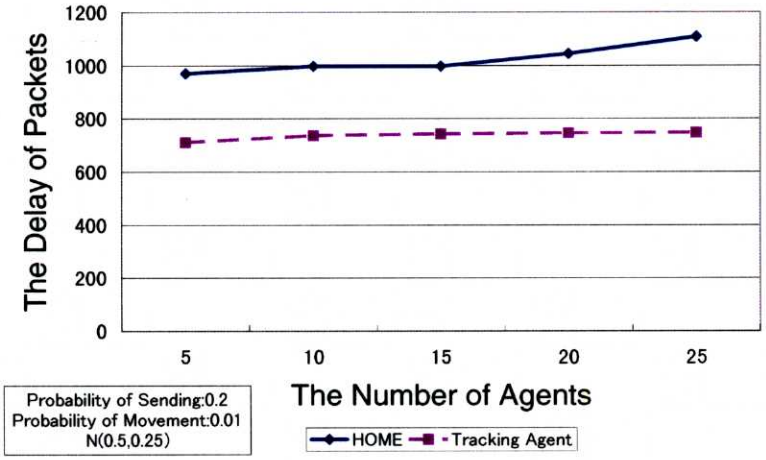


Figure 3.8: モーバイルエージェントの分布が偏っている場合のパケット遅延（1）

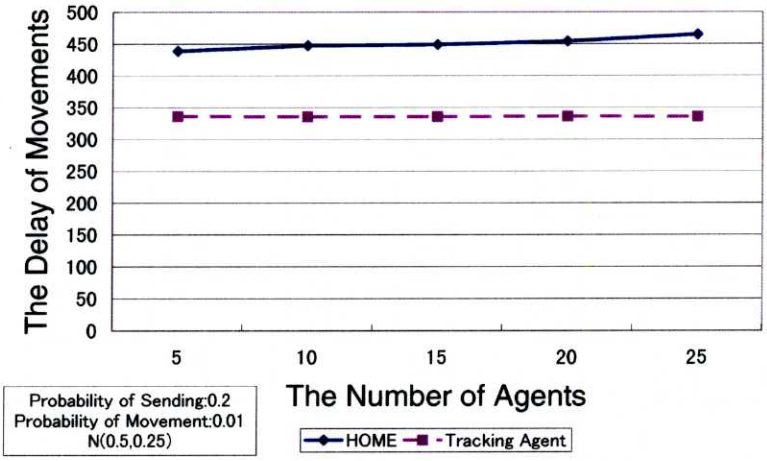


Figure 3.9: モーバイルエージェントの分布が偏っている場合の移動に要する時間（1）

図 3.8、3.9 はモーバイルエージェントの個数に対するパケットの遅延、及び移動に要する時間を表している。モーバイルエージェントの分布は $N(0.5, 0.25)$ に従う正規分布乱数を用いてネットワークの中心に偏らせている。これによってモーバイルエージェントは格子状ネットワークの

中心付近の 1/4 程度の面積に部分に偏在することになる。また、単位時間Δt における各モーバイルエージェントのパケット送信確率は 0.2、移動確率は 0.01 である。

トラッキングエージェントを用いると HOME の場合に比べてパケット遅延、移動に要する時間とも 3 割程度減少していることがわかる。

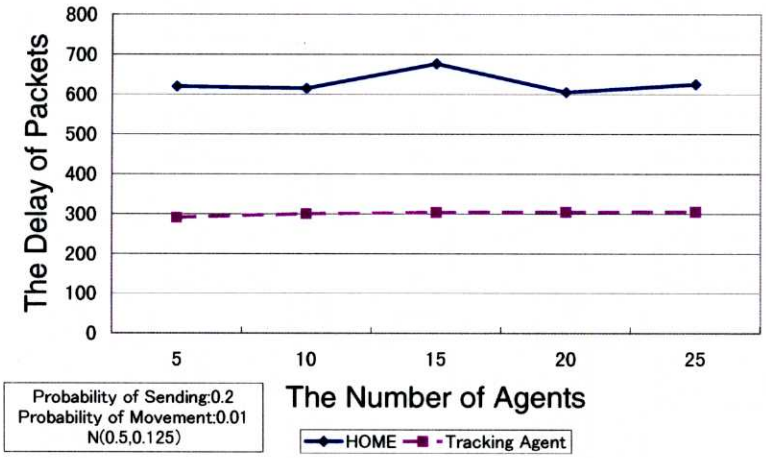


Figure 3.10: モーバイルエージェントの分布が偏っている場合のパケット遅延（2）

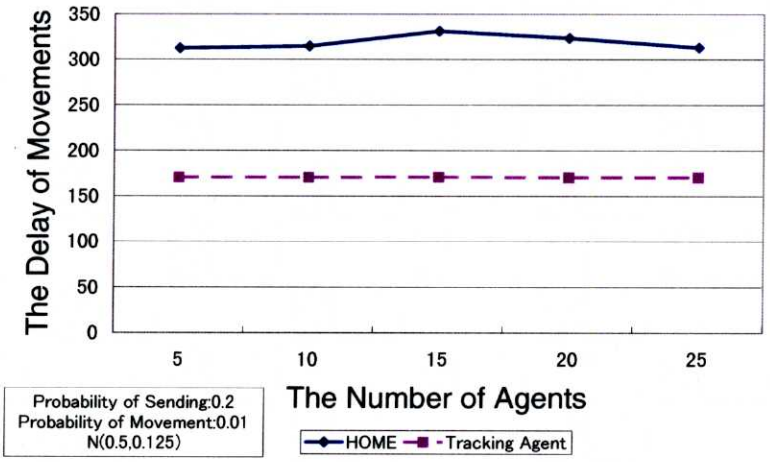


Figure 3.11: モーバイルエージェントの分布が偏っている場合の移動に要する時間（2）

一方、図 3.10、3.11はモーバイルエージェントの分布を $N(0.5, 0.125)$ に従う正規分布乱数を用いてさらに偏らせた場合の、モーバイルエージェントの個数に対するパケットの遅延、及び移動に要する時間を表している。単位時間Δt における各モーバイルエージェントのパケット送信確

率は 0.2、移動確率は 0.01 である。

この場合はモーバイルエージェントは格子状ネットワークの中心付近の 1/16 程度の面積に部分に偏在することになり、グラフからパケット遅延、移動に要する時間とも約 1/2 になることがわかる。

本シミュレーションによって、トラッキングエージェントを用いた方式はブロードキャスト方式より遅延、及びパケット送信数ともに少ない通信ができる事が示された。特に協調するエージェントの数が増加するにつれて、ブロードキャスト方式は遅延が大きいばかりではなく、ネットワークに対して非常に負担をかける事になる。本シミュレーションにおいてはネットワーク飽和しないものと仮定したが、実際のネットワークにおいてブロードキャスト方式を用いるのは適当ではない。また、HOME を用いる方法と比較した場合においてはモーバイルエージェントの分布の偏りがある場合にはトラッキングエージェントを用いるとパケット遅延、移動に要する時間を低減する効果があることが示された。

3.6 むすび

多岐に渡るエージェントの用法の中で、本論文では複数のモーバイルエージェントが協調する環境に着目し、ネットワーク上を動的に移動するモーバイルエージェントのエージェント間通信を支援するためにトラッキングエージェントを提案した。

トラッキングエージェントはモーバイルエージェント間の通信を仲介し、モーバイルエージェント同士では移動を意識させない協調関係を提供することが可能である。さらに必要な時にだけ生成することにより、ネットワーク上の資源を有効に活用することができる。また、トラッキングエージェント自身も必要に応じて移動することにより経路の冗長度を下げることが期待できる。

計算機シュミレーションによって送信パケット数、メッセージ遅延およびのモーバイルエージェントの移動に要する時間を評価した。トラッキングエージェント方式はモーバイルエージェントの数が増加してもブロードキャスト方式ほどパケット数が増加することはない。また、モーバイルエージェントの分布に偏りを持たせた場合には、HOME を用いる方式と比較してメッセージ遅延、および移動に要する時間とも大きく低減されることが示された。

本シミュレーションにおいてはトラッキングエージェントの移動先の候補をモーバイルエージェントの重心から求めた。しかしながら実際のネットワーク上では最適な重心を求める事は困難である。各モーバイルエージェントの通信量を考慮した重み付けがさらに必要である事を考慮すると、一つの簡便な方法として最も通信量の多いモーバイルエージェントから最も近いところに存在するマザーエージェントの存在するホストを、トラッキングエージェントの移動先の候補とする方法などが考えられよう。

当研究室では JAVA を用いてトラッキングエージェントのプロトタイプの実装を行い、その報告を [10] に述べている。

Bibliography

[1] Leonardo Chiariglione. Foundation for intelligent physical agents. http://drogo.cselt.stet.it/fipa/.

[2] GMD FOKUS. Mobile Agent System Interoperability Facilities Specification – Joint OMG
Submission, Mar. 1998. ftp://ftp.omg.org/pub/docs/orbos/97-10-05.ps,http://www.camb.open%group.org/RI/MAF/drafts/draft12/draft12.book.ps.gz.

[3] Object Management Group. OMG Homepage. http://www.omg.org/.

[4] Christian Huitema. *IPv6 : The New Internet Protocol.* Prentice Hall PTR.., Jan. 1996. WIDE プロジェクト IPv6 分科会監訳.

[5] C. Perkins. IP Mobility Support. RFC 2002, Oct. 1996.

[6] 飯田一朗. パーソナル移動通信のためのソフトウェア. 電子情報通信学会誌, Vol. 78, No. 2, pp. 150–155, Feb 1995.

[7] 飯田一朗, 西ヶ谷岳. モーバイルエージェントとネットワーク. 情報処理学会誌, Vol. 38, No. 1, pp. 17–23, Jan 1997.

[8] 松下温, 中川正雄, 寺岡文男, 中島達夫, 宮澤正幸, 若尾正義, 小澤和幸, 軒野仁孝. ワイヤレス LAN アーキテクチャ. 分散協調メディアシリーズ, No. 7. 共立出版, 1996.

[9] 國頭吾郎, 奥村恭弘, 相澤清晴, 羽鳥光俊. 協調エージェント間通信のための tracking agent に関する 検討. 信学技報 IN97-20,CS97-1,MVE97-1, 電子情報通信学会, Apr 1997.

[10] 中川智尋, 國頭吾郎, 相澤清晴, 羽鳥光俊. モバイルエージェント間の通信の為の Tracking Agent の Java による実装. 電子情報通信学会全国大会, pp. B–7–46, Mar 1998.

[11] 小島直人, 新谷虎松. 複数エージェントシステムにおける通信の効率化. 情報処理学会第 50 回全国大会講演論文集, pp. 2–167, 1995.

[12] 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, Vol. CS95-133, , Nov 1995.

Chapter 4

モバイルエージェント言語とトラッキングエージェントの実装

モバイルエージェントの実現のためにいろいろなエージェントプラットホームが開発されている。ここではいくつかのエージェントプラットフォームを紹介する。その後 AgentTcl を用いたトラッキングエージェントをの実装について述べる。

4.1 モバイルエージェントのための必要要件

ここではエージェントを以下のようなプロセスであると定義する (図 4.1)。

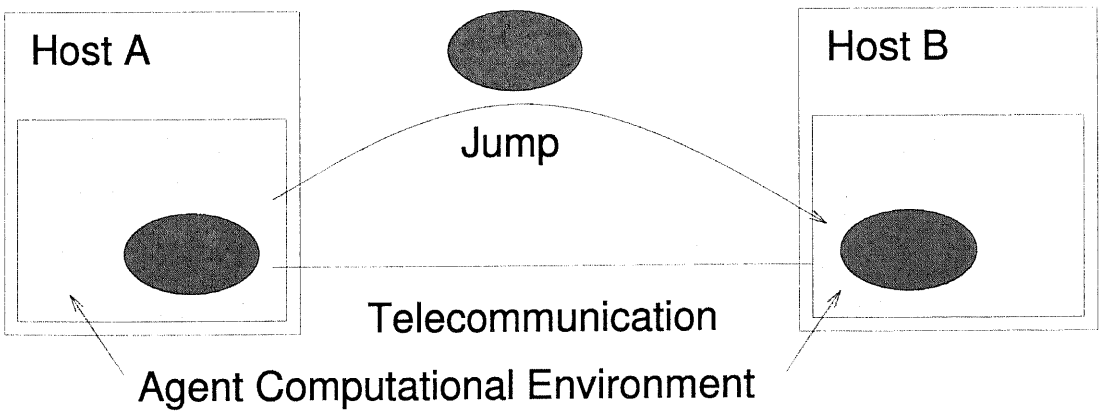


Figure 4.1: エージェント

- エージェント実行環境で実行可能
- 離れたエージェント同士で通信できる

- リモートにあるエージェント実行環境へ移動可能

以下エージェントプラットフォームの実現という観点から、このようなモバイルエージェントの特徴をより詳しく述べる。

4.1.1 ノード間でのプログラムの移動機能

モバイルエージェントは、プログラムの処理の途中であってもその処理を停止し、移動先のホストで停止した部分から処理を再開することができる必要がある。モバイルエージェントの技術的なメリットは次の通りである。

- エージェント型の通信によるインタラクショントラヒックの減少
- オフラインオペレーション

  エージェントがサーバ側で処理を行なっている間、サーバークライアント間の回線を切断できる。
- 低能力クライアントのサポート

  処理をサーバ側でできるので、クライアントの記憶容量、処理能力が低くても良い。
- サービスクライアントソフト配布の効率化
- Semantic Routing

  エージェントはお互い ACL(Agent Comunication Language) を用いて情報をやり取りし、自分が持つ、あるいは他のエージェントが持つインデックスを頼りにクライアントのリクエストをサーバに送信する。
- 規模透過性

  システムやアプリケーションの規模の変更がシステムの構成やアプリケーションのアルゴリズムを変更せずに、またシステムの機能を停止せずに可能である。
- オーバヘッドが小さいセキュア・トランザクション処理

  メッセージ通信と同様、再送による遅延の影響が RPC 程深刻ではない。スケーラビリティに関してエージェントは RPC より有利だが、その分遅延も大きい。

4.1.2 内部状態の管理

エージェントは一般に自分の行動に影響を与える複数の内部状態を持っている。内部状態は、エージェント内部で動作するプログラムによって動的に書き換えられ、書き換えられた内部状態がエージェントのその後の動作に条件や制約を与える。この内部状態のデータベースを管理するインターフェースがあるとよい。

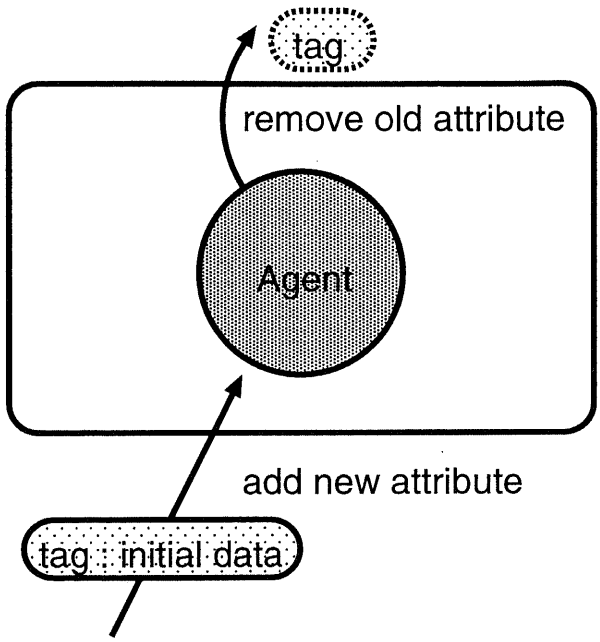


Figure 4.2: 内部状態の管理

セキュリティの観点からこの内部状態が外部から直接書き換えられるようにするとは考えにくい。しかしながら、エージェント同士が従属関係にある場合、親のエージェントがサブ・エージェントの動作を規定するために親エージェントにだけ直接内部状態の変更を許可する場合もありうる。

4.1.3　エージェント間の交渉（ネゴシエーション）

各エージェント間で協調して問題解決にあたる場合などには、エージェント間の通信機能が必要である。エージェント間の情報交換のプロトコルを以下のように分類する。

4.1.3.1　エージェント通信言語 ACL(Agent Communication Language)

異なるエージェントが同一のシステム、同一の知識を持っているとは限らないことから、エージェント間のやり取りは共通の言語を用いる必要がある。ここで、ACL は通信のロジック、エージェントの動作を規定するものである。こうした言語に KQML[12](Knowledge Query and Manupulation Language) や AgenTalk[1] がある。

KQML は DARPA(Defense Advanced Research Project Agency) による知識共有プロジェクトにおいて開発された言語で、分散された知識システム間での知識共有や再利用を行なうために考えられたものである。AgenTalk はプロトコルを記述するための言語で、拡張有限状態オー

トマトンとしてのプロトコル表現であり、継承、並列処理といった特徴を持つ。

4.1.3.2　知識交換用言語

知識交換用言語として、KIF[6](Knowledge Interchange Format) がある。KIF は知識ベースシステム間の情報交換用に設計された言語であり、構文と意味は統合を含む一階述語論理の拡張となっている。

KIF では知識ベースの手続き的な内容を捨て、宣言的な所だけを知識表現言語に独立しようとしている。また、KIF は計算機のための言語であり、人間に読み易いような配慮はなされていない。

4.1.3.3　オントロジー記述言語

エージェント間の相互作用を成立させるためには、エージェント間のメッセージに含まれる用語に関する同意が必要である。このためには、各エージェント間で利用できる共通オントロジー(基本概念の体系) が必要である。ここで扱うオントロジー記述言語とは、用語の集合と、用語間の論理的制約を記述するための言語である。オントロジー記述言語に Ontolingua[15] がある。

4.1.4　メッセージパッシング: 同期、非同期

一般にエージェント間でやり取りされるメッセージは、非同期なメッセージ通信をベースにしていることが多い。一方、分散オブジェクト指向モデル (例えば CORBA、OLE など) では、RPC を基本とした同期通信が一般的である。RPC モデルでは、関数呼びだしとしてのメッセージを他のエージェントに送信したあと、受信したエージェントが戻り値としての返答メッセージを返すまで、送信側エージェントのスレッドはブロックされるため、処理の並列性が損なわれてしまうという欠点がある。しかしながら、他のエージェントの返答メッセージの内容によって動作を変えるような振舞いを記述する場合には、RPC モデルの方が同一の action 定義内に他のエージェントとのインタラクションをシーケンシャルに記述できるため、従来の高級言語と同じようにコーディングが容易で可読性が良くなる。同期／非同期には双方とも異なる利点があり、状況に応じて使い分けることができることが望ましい。

4.1.5　リフレクション

エージェントの振舞いを規定する action の定義をランタイムに変更することを可能にする仕組み (図 4.3) をリフレクションといい、システムに柔軟性を与えることができる。リフレクションによって次のようなメリットが得られる。

- ランタイム環境でのソフトウェア保持やアップデート
- 新しいポリシーを埋め込んだソフトウェアのカスタマイズ

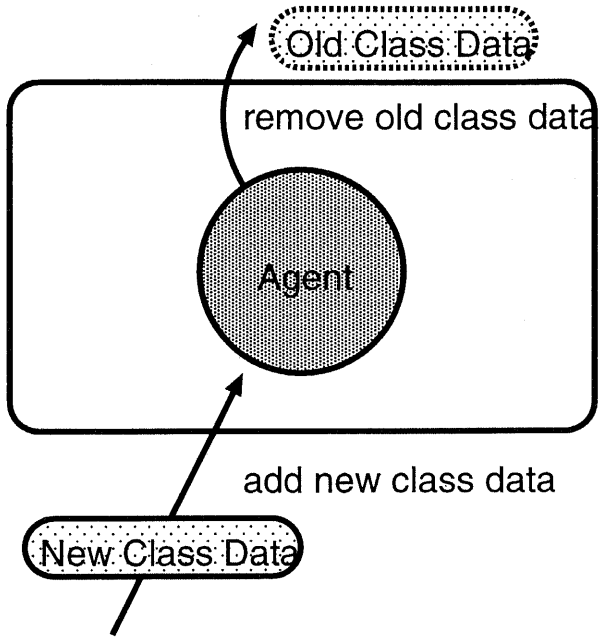


Figure 4.3: リフレクション

- 実行コードのスリム化。長い運用期間の中でめったに使われないプログラムや、実行環境に依存して利用されない不要なプログラムによりアプリケーションが肥大化するのを防ぐ

リフレクションは、モバイルコードを利用したアプリケーションには強力な feature になると思われる。コードの転送は RPC に比べて通信のオーバヘッドが大きい。しかし、移動先の環境に必要なコードだけを転送するようにすれば、移動のオーバヘッドを削減できる可能性があり、動的な負荷分散やエージェント間通信の応答速度の改善にダイナミックに対応できるようになる。

リフレクションをサポートする場合にランタイム環境では次のような問題が起こり得る。

- 削除されたメッセージインタフェースに対してメッセージが送られてきた場合の対処
- メッセージに添付される引数の制約が変更になった場合の対処
- アクションが動作中にそのアクションの定義に対して変更要求を受けた場合の対処

4.1.6　リモートエバリュエーション

一般に、メッセージに対応して起動されるアクションは、予め相手のエージェントに用意されている必要がある。リモートエバリュエーションとは、アクションを定義したプログラムを直接送り込み、相手の実行環境上でで実行させることができることをいう。これには相手が要求した

ときにコードを渡して相手に実行させるダウンロード型 (図 4.5) と、非同期に相手にコードを送りつけて相手に実行させるアップロード型 (図 4.4) と、2 つのモデルがある。リモートエバリュエーションは、以下の場合で有効である。

- デバッグツールやメンテナンスツールのように、常時使われるコードではなく、タスクの終了後は他のアクションから参照される必要のない場合、通常はエージェントのコードを小さくできる。
- セキュリティアルゴリズムを含むコードで、アルゴリズムを毎回変更して実行させたい場合
- バッチ処理などのように、特定のエージェントに対して、連続して複数の要求を送信する場合、エージェント間のメッセージトラヒックを減らす効果がある。

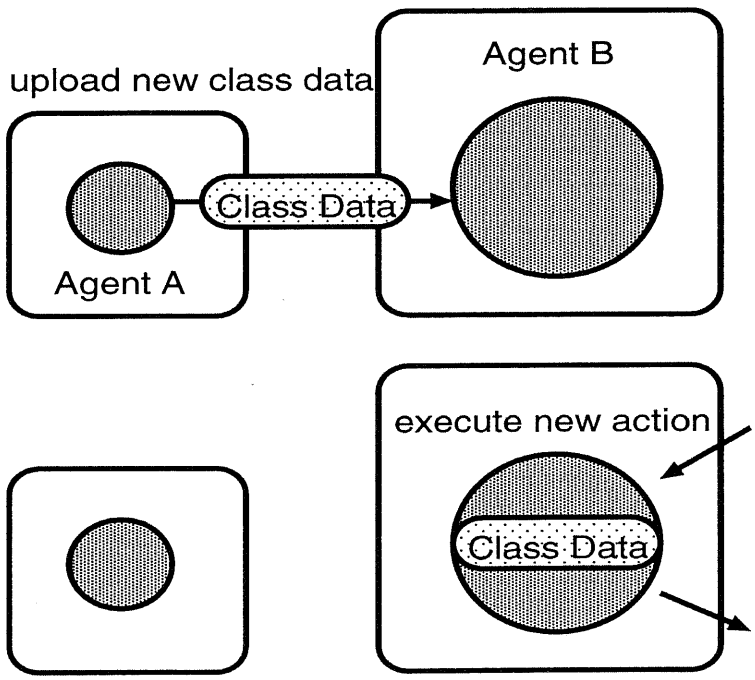


Figure 4.4: アップロード型

ダウンロード型、アップロード型のいずれの場合であっても、プログラムにバグがあったり、悪意を持ったプログラム作者、あるいはクラッカーなどによってそのプログラムに破壊的なプログラムが仕込まれたりすると被害が広範囲に及ぶ可能性があるので、セキュリティ機能が不可欠である。

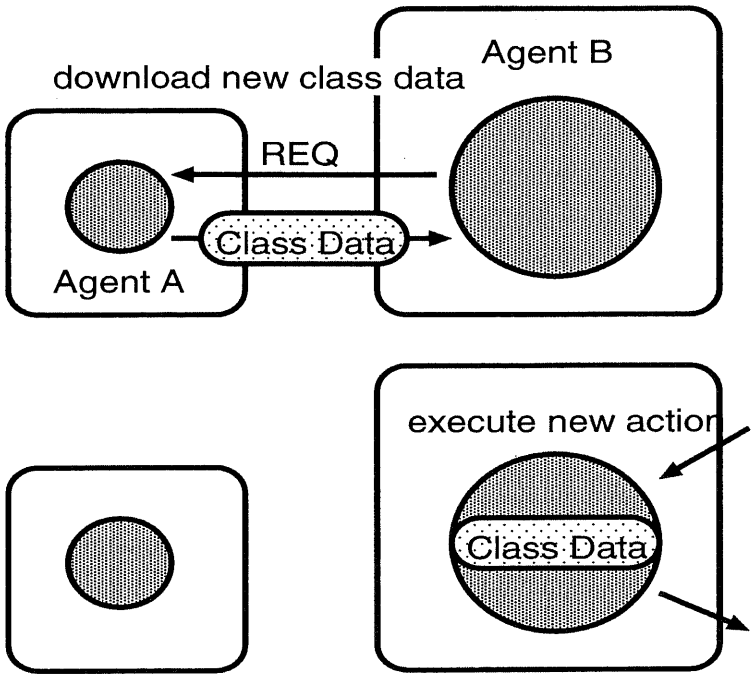


Figure 4.5: ダウンロード型

4.1.7 セキュリティ

分散オブジェクト環境では、エージェントのプログラムが実行ノード上で偶然あるいは故意に、ノード上のリソースに対し被害を及ぼさないことが求められる。リソースのアクセス権を無視してデータを取り出したり、故意に改竄や破壊をしない機構が求められる。

4.2 モバイルエージェント言語

これまでに述べたような条件を満たすべく、いろいろな言語が開発されている。

Java 系	Aglets, Kafka, Voyager, Odyssey, Mole etc.
Tcl/Tk 系	TACOMA, Safe-Tcl, D'Agents etc.
その他	Telescript,Obliq, April, Python, M0, Tycoon, Facile etc.

Table 4.1: エージェント言語

これらの言語のなかから、いくつかのものについて簡単に説明する。なお、製品のバージョン

は全て 1999 年 1 月現在のものである。

4.2.1 Java 系

Java は多様な計算機環境で利用できるインタプリタ言語として開発された言語であって、モバイルエージェントの実現を意図したものではない。しかし、アプレットをリモートノードで実行させるということから、モバイルエージェント言語として分類することもできる。

JDK 1.1 のリリースにより標準で Java RMI(Remote Method Invocation) の分散環境でのコミュニケーション機能を使用することが可能になり、エージェント記述言語としての Java は強力な地位を固めつつある。

モバイルエージェントの老舗ともいえる General Magic 社も現在は Java ベースの移動エージェント用フレームワーク Odyssey のβ2 をインターネット上で公開している。このようにベンダーが製品化を前提に開発している移動エージェント用フレームワークは最近急増している。

4.2.1.1 Aglets Software Development Kit

IBM Tokyo Research Laboratory が開発 [2]。現在は ver 1.1 Beta と ver 1.0.3 である。

Aglets はモバイルエージェントとして Aglet クラスを持ち、これらの保守管理を AgletContext クラスが行なう。

Transfer Protocol は ATP(Agent Transfer Protocol: OMG[9] の Mobile Agent Facility のプロポーサルとして提出された) を利用している。アクセスコントロール、リフレクション機能はない。メッセージパッシングについては、同期、非同期、戻り値なしの 3 種類から選択できる。リモートエバリュエーションについては、モバイルエージェントそのものによって実現している。

分散しているエージェントサーバについては何らかの形でアドレスを取得する必要があり、分散ディレクトリは考えていない。

4.2.1.2 Voyager

ObjectSpace 社が開発 [7]。現在 ver 2.0.1.

Voyager の VAgent クラスは Agent クラスのサブクラスを、VObjact クラスは一般の Java クラスを vcc(Virtual Class Compiler：プリコンパイラ。このツールで Virtual クラスを作成する。) でコンパイルされて生成されるクラスの親クラスとなる。

Voyager は独自の ORB(Object Request Broker) を持ち、これにより異なる VM 上で RMI より 1.2~2 倍高速であると報告されている。アクセスコントロール、リフレクション機能は特に備えていない。メッセージパッシングにおいては、Messanger のサブクラスとして Sync(同期型), Future(非同期型), OneWay(戻り値なし) の 3 つのクラスを用意している。リモートエバリュエーションに関しては、モバイルエージェントそのものの他に、Virtual クラスを用いた RPC を使う方法がある。分散ディレクトリに関しては、将来的にサポートするらしいが、まだよくわからない。

4.2.1.3 Odyssey

General Magic 社が開発 [8]。 ver1.0 Beta 2 が公開されていた。

Odyssey の OdysseyProcess クラスを親クラスとして、Place と Agent クラスを作る。これらのクラスは Abstract クラスなので必ずユーザはサブクラスを作成しなければならない。JVM(Java Virtual Machine) が立ち上がる時、Odyssey のエージェントシステムを起動するときの BootPlace を指定し、Place の設定を行なう。

Transfer Protocol は、RMI, DCOM, IIOP(Vigigenic の VisiBroker for Java を利用) のいずれかを選択できるようになっており、将来の動向にいちはやく対応しようとしている。アクセスコントロール、リフレクション機能については特に備えていない。メッセージパッシングについて、Odyssey では全てモバイルエージェント間の対話によって通信を行なうので、特別な機能は用意されていない。リモートエバリュエーションについては、モバイルエージェントそのものによって実現している。分散ディレクトリに関しては考えていない。

4.2.2 Tcl/Tk 系

Tcl/Tk の開発拠点が sunlab に移動したことによって、SunScript と呼ばれるようになる。Lucent Technology、SCO、DynaWeb などが主要なパートナーとなっている。現在は Tcl/Tk8.0p1 と、そのツールキット TclPro 1.2Beta が発売されている [10]。

Netscape Navigator 向けに plug-in がリリースされており、WWW Browser 上のスクリプト言語として利用することも可能である。また、Jacl という、SunScript を 100% Pure Java でリインプリメントしたものもある [5]。

UNIX、Windows、Macintosh 向けのリリースが存在し、"Write Once, Run Anyware" を実現している。

4.2.2.1 TACOMA(Troms And COrnell Moving Agents)

Troms Univ.(Norway) と Cornell Univ.(USA.) が開発。現在のバージョンは 1.2[11]。

Tcl スクリプトから、他のホストで動いている Tcl スクリプトを呼び出すプリミティブを追加し、拡張した。これにより、リモートのホストにスクリプトコードと初期データを送信することができる。

Tcl ではコマンドの実体が見つからないと、unknown コマンドが実行される。デフォルトでは、ローカルのパッケージとライブラリを探し、見つければこれをロードし、実行する。unknown コマンドを再定義することで、振舞いを変更することができる。

モバイルエージェントは、UNIX のプロセスとして実装され、その実行環境は UNIX の OS と実行時のサポートで実現されている。異なる実行環境上のエージェントが他方のコードを実行する場合、briefcase というデータ構造にコードと初期化データを納め、送信する。受信側では、受けとったコードから新しいエージェントを生成する。この新しく作成されたエージェントは、呼び出し側のエージェントから受けとった briefcase のデータにアクセスする。このデータは受け

とった実行環境の一部となる。送られたコードはどのリソースにもバインドされないため、データスペースの制御は必要ない。

セキュリティに関しては、あるエージェント実行環境では、特定のホストからしかエージェントを受けとらないようになっている。また、アクセスコントロールは考えておらず、複雑なモバイルコードアプリケーションには不向きである。

4.2.2.2 D'Agents

Darthmouth Univ. において開発 [4]。Agent Tcl と呼ばれていたものであり、現在のバージョンは 2.0 である。

Tcl のインタプリタを拡張し、モバイルエージェントをサポートしている。動的リンクに関しては TACOMA と同様に、unknown コマンドを利用。

エージェントは、UNIX 上で動くインタプリタのプロセスとして実現されている。その実行環境は OS の機能と、名前空間やエージェント間通信をサポートする実行時環境で実現されていて、リモートで新しくエージェントを作り実行することも、もとのエージェントを移動させて実行することもできる。

4.2.3 その他

4.2.3.1 Telescript

米 General Magic 社 が開発した Telescript は モバイルエージェントの概念を明確に打ちだし、モバイルエージェントを世に知らしめた言語である。ネットワーク上の計算機資源（ノード）上で、様々なエージェントがノードを渡り歩いて処理を行なうことにより、ノード間の Electronic Market Place を実現する目的で構築された。構文は C++に似ており、オブジェクト指向言語である。TSE (TeleScript Engine) と呼ばれるインタプリタにより実行されるが、この TSE はプラットフォームに依存しない独立な一種の仮想ＯＳであり、マルチプロセスやメモリ管理をサポートする。

Java がダウンロード指向のモーバイルコード言語だとすると、Telescript はアップロード指向のモーバイル言語であるといってよい。つまり、一般ユーザーのクライアントマシンや他のサーバーマシンから公共のサーバーマシンへとプログラムすなわちモーバイルエージェントが転送され実行される。

Place という実行環境上で、時にはそれを渡り歩くことでエージェントは処理を行なう。ある Place 上で鉢合わせたエージェント同士は、会話することができる (meeting)。アクセスコントロールについて Permits 型 (アクセス制御機構) を持ち、実行可能命令や使用するリソースを制限できる。スコープに関しては、スレッドとスレッドが所有するオブジェクトの間で ownership が定義され、名前解決はまずそこで行われる。もし解決できなければ、上位の place で解決を試みる。こうして、place を再帰的に最上位までたどり、 名前解決を行う。

現在 General Magic 社は、Telescript を中断し、Java ベースのフレームワークを発表している。

4.2.3.2 April

富士通研究所が開発 [3]。

April のプロセスには `local-process-name@host-name` という形式をしたグローバルネットワーク上でユニークに識別できる名前が付けられる。host-name はフルドメイン形式のホスト名で、local-process-name はユーザが指定可能である。指定しなかった場合はシステムが自動で適当な名前をつける。このプロセス名を使うことにより、下位層のネットワークプロトコルを意識せずプロセスを指定することが可能である。同一ホスト内でも遠隔ホスト間でも透過的にプロセスを指定し、メッセージ送信、プロセス状態チェック、プロセス終了待ちを行なうことができる。

メッセージの送信は message >> receiver で行ない、メッセージの受信は pattern -> statement で行なう。pattern にマッチする message が送られてくると statement が実行される。この時マッチしたデータが pattern に含まれる変数にバインドされ、statement のスコープ内で参照できる。pattern -> statement は複数並べて書くことができ、そのうちの 1 つの pattern にマッチすると対応する statement が実行される。このパターンマッチング機構により、メッセージを手続き的に解析することなく、また複数のパターンを使ったメッセージ受信を効率的に行なうことが可能である。

April では関数/手続きおよびその集合体であるモジュールを first-class object とし、他のデータと同じように取り扱う。即ち、実行コードも他のデータと同じようにメッセージとして、ネットワーク上にある任意のプロセスに送り、そこで実行することができる。 よって、受けとった側で新しいプロセスが生成可能で、モバイルエージェントを可能としている。また、ACL に KQML を採用している。

4.3 Agent Tcl を用いたトラッキングエージェントの実装

4.3.1 Agent Tcl の開発目的とその構造

Agent Tcl は Dartmouth 大学において

- 端末間移動を任意時点で行なえる 1 つの命令にする
- エージェント間通信を提供する
- 転送メカニズムをサポートする
- 一般的 Unix プラットホームでの起動が可能で Unix 以外のプラットホームへできるだけ簡単に移動できる
- インターネット上での効果的なセキュリティ機構を提供する

- 公共領域で利用できる

という目的で、TCL (Tool Common Language) を基に開発された Transportable Agent 言語である。

Agent Tcl はトランスポート層の上に 4 階層で構成されている。トランスポート層として TCP を用いているが、Email をトランスポート層として用いることもできる。トランスポート層の上で各ホスト上で起動される Agent Tcl Server Engine が動作する。Agent Tcl Engine の上でインタプリタが走り、Agent はさらにその上の層で動作する。

また、インタプリタはエージェントが悪意のある行為をすることを防ぐセキュリティモジュール、起動中のエージェントの内部状態を把握し、また元に戻すためのステイトモジュールを含み、マイグレーション、コミュニケーションそしてチェックポインティングに関する API を持つ。

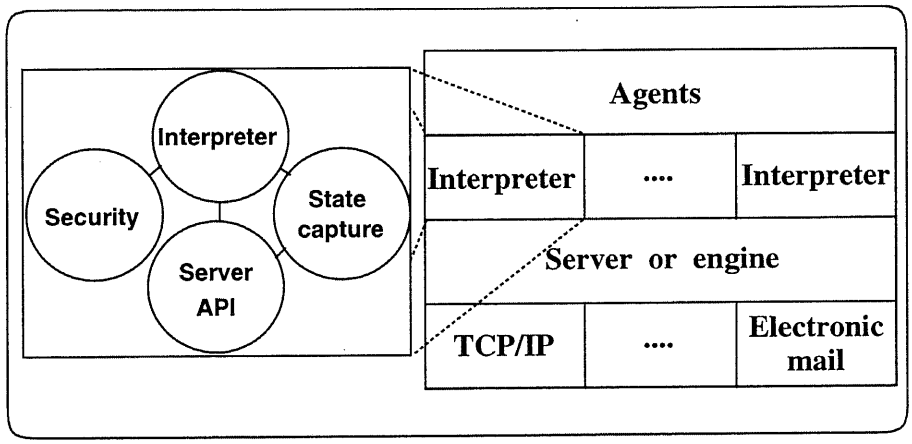


Figure 4.6: Agent Tcl の構造

モバイルエージェントの機能としては

- 端末間移動
- メッセージ送信
- ダイレクトコネクション
- 基本的なセキュリティ

を提供している。

Agent と サーバとは図 4.7のような関係になっている。

サーバは、2 つのプロセスが協調して実行される。プロセスの 1 つは入ってくるエージェント、メッセージそしてリクエストに対する Unix ソケットを監視するソケットウォッチャーであり、も

う 1 つは端末上で起動しているエージェンの追跡、相手のエージェントがメッセージを受け入れるまでのメッセージの縮小を行なうエージェントテーブラーである。

エージェントはコマンドを提供する修正された Tcl コアと Tcl の拡張から成る。内部的にはどのコマンドもサーバとコンタクトをとり、エージェント、メッセージリクエストを転送し承認を待つために API を用いる。

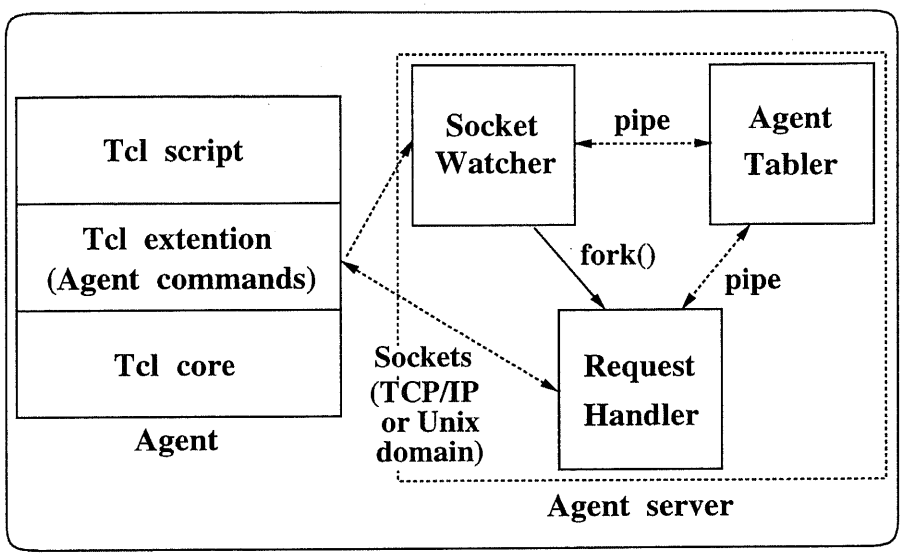


Figure 4.7: alpha release の構造

4.3.2　エージェント間通信コマンド

Agent Tcl には 2 種類のエージェント間通信コマンドが与えられている。

一つは、agent_send、agent_event という相手のエージェントのホスト名又はホスト IP とエージェントの名前又は ID 指定によりメッセージ送信を行なうものであり、agent_event はタグを指定することでメッセージをイベントとして送信することが可能である。これは、受信側が mask コマンドを用いて特定のタグに対してイベントハンドラーを与えることができる。つまり、受信側は特定のタグを持つメッセージを特定のイベントハンドラーにより自動処理することができるという利点がある。

もう一つは agent_meet によりダイレクトコネクションを張った後、そのコネクションにそってメッセージ送信を行なうものである。ここで、ダイレクトコネクションは以下の手順（図 4.8）で張られる。

1. エージェント A が B にコネクションリクエストを行なう。

2. エージェント B はリクエストを受け入れるか拒否するかを判断し、拒否の場合コネクションは張られない。

3. エージェント B が要請を受け入れた場合コネクションが張られ、両方のエージェントにそれぞれソケット番号が与えられる。

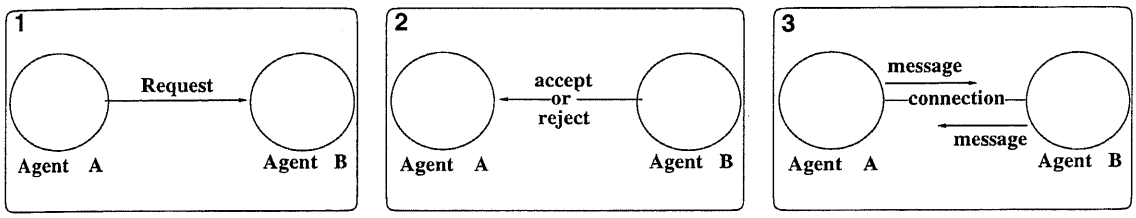


Figure 4.8: ダイレクトコネクション

また、ソケット番号はこのコネクションによるメッセージ送受信に用いられるものであり、コネクションの識別番号のようなものである。

4.3.3 メッセージ送受信スクリプトの処理時間の比較

2つの簡単なエージェント（以下A、B）を考える。Bはメッセージを受信したら即座にAに返答メッセージを送信する。AはBにメッセージを送信し、その返答メッセージを受信する（図4.9、図4.10）というスクリプトの処理時間を計測するものである。

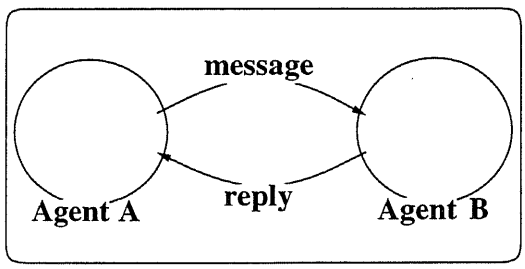


Figure 4.9: 測定1、2

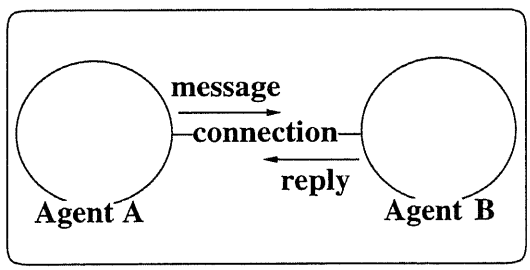


Figure 4.10: 測定3

この処理時間の測定を 1.(agent_send, agent_receive) 2.(agent_event, agent_getevent) 3.(tcpip_write, tcpip_read) の3種のメッセージ送受信コマンドに対して行なった。

表 4.2に、結果を示す。但し、3の場合はダイレクトコネクションがすでに張られているとする。この結果からも明らかなように、ダイレクトコネクションを張ったメッセージ送受信が他の2つと比べ断然に効率的であることがわかる。

4.4 実装環境

実装は、東京大学電気電子情報学科 羽鳥・相澤研究室内の端末において行なった。

Table 4.2: 平均処理時間

コマンド	Connection	平均処理時間
1.(agent_send,agent_receive)	無し	58msec
2.(agent_event,agent_getevent)	無し	62msec
3.(tcpip_write,tcpip_read)	有り	5msec

まず、実装された通信体系を簡略化して表したものを図 4.11に示す。

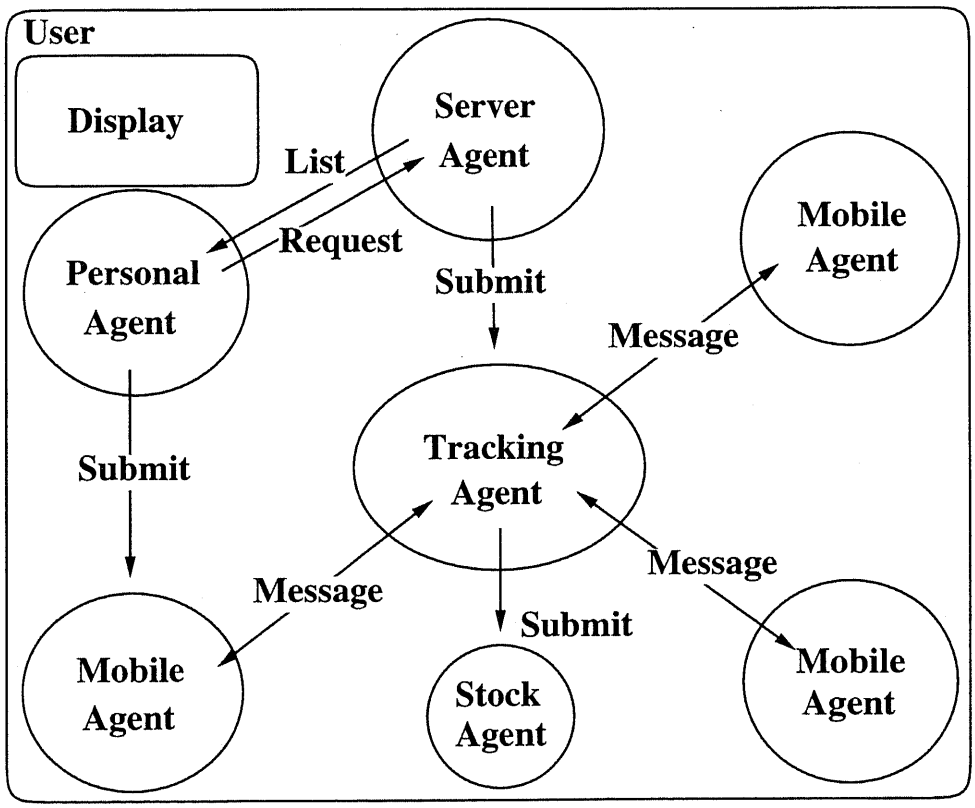


Figure 4.11: 実装体系

本実装において、Server Agent は任意のホストに配置され、このホスト名はユーザに既知であるとする。またユーザの元に 2 つのエージェント（Personal Agent、Mobile Agent）を配置した。まず Personal Agent をユーザの元に固定配置し、エージェント-ユーザ間ユーザインター

フェースを実現し、次に Personal Agent の子供として移動可能な Mobile Agent を生成した。

また、Tracking Agent は Personal Agent からの要求により Server Agent の子供として、Stock Agent は Mobile Agent からの要求により Tracking Agent の子供として生成した。

4.5 実装手法とインターフェース

4.5.1 Server Agent の配置

まず最初に、任意のホストに Server Agent が配置されなければならない。Server Agent は Tracking Agent の生成、現在登録可能な Tracking Agent リスト、また登録している Mobile Agent リストの管理、これらのリストの送信を行なう。Server Agent は図 4.12に示す Server Display を通して制御することが可能である。

必要であれば Value ボタンをクリックした後、現れる Value1 Display （図 4.14）によって、Server Agent、Tracking Agent の持つ必要な値を知ることが可能である。ただし、Tracking Agent の値が必要な場合は、 Value2 Display （図 4.13）によって Tracking Agent を指定する必要がある。

また Server Display の Exit ボタンをクリックすることにより Server Agent のスクリプトを任意時点で終了させることが可能である。

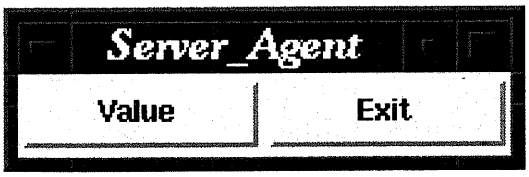


Figure 4.12: Server Display

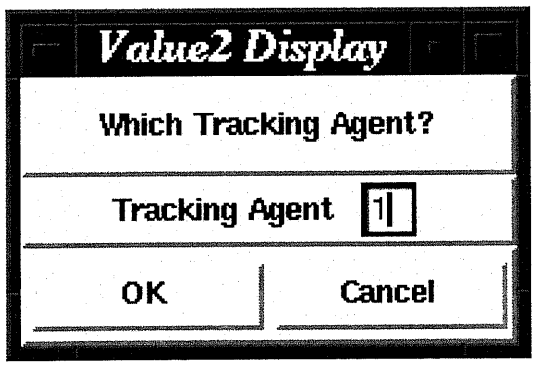


Figure 4.13: Server Value2 Display

4.5.2 スタートアップ

ユーザはまず、図 4.15に示す Start Display に Server Agent を配置したホスト名、Mobile Agent 名を入力する。Personal Agent は入力に従って、同一の名前を持つエージェントが同一ホストに存在しないか、Server Agent は入力されたホストで起動されているかの確認を行ない、問題無い場合は Start Display が消え、Main Display （図 4.16）が現れる。この間に Mobile Agent に入力された名前が付けられる。

Main Display では Mobile Agent の名前、現在の未読メッセージの数が自動的に表示され

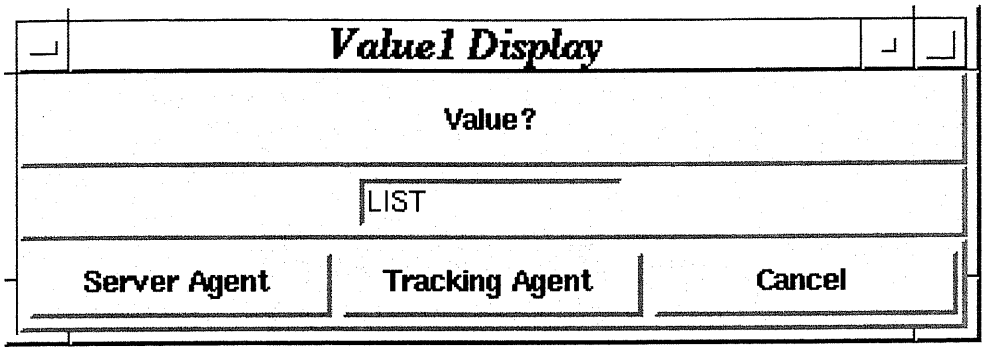


Figure 4.14: Server Value1 Display

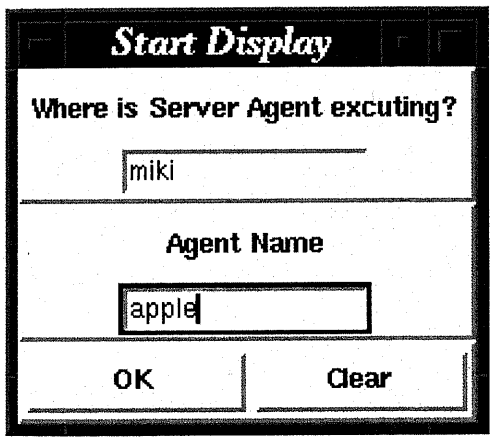


Figure 4.15: Start Display

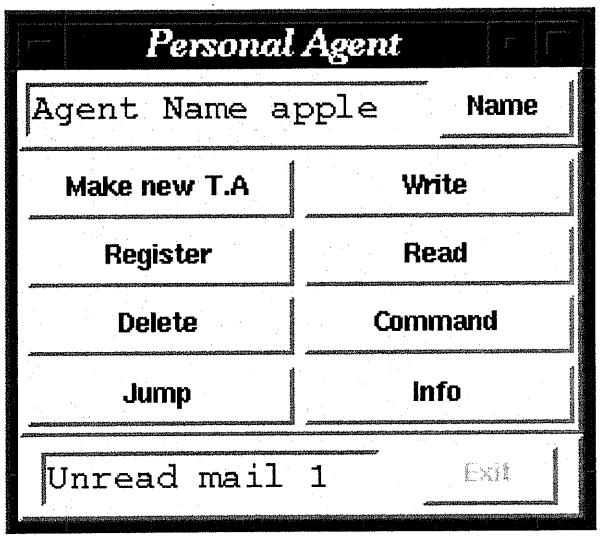


Figure 4.16: Main Display

る。次に Main Display 以外のウインドウが出力されている場合、Mobile Agent が いずれかの Tracking Agent にまだ登録している場合は Main Display の Exit ボタンを使用不可能にしてある。登録されている Mobile Agent の数が 0 になったところで Tracking Agent は自動消滅するよう設計されているため、Mobile Agent が登録したまま削除要請を出す以前に終了されると、Tracking Agent が自動消滅できず何もしないエージェントがネットワーク上に残ることになり、ネットワーク資源の無駄であるからである。

4.5.3 名前

Mobile Agent の名前は任意時点で変更が可能である。Main Display において Name ボタンをクリックすると Name Display（図 4.17）が現れる。名前を変更したい場合は、Name Display に新たな名前を入力し OK ボタンをクリックすればよい。また、名前の変更が行なわれた場合 Main Display の名前表示も自動的に変更される。

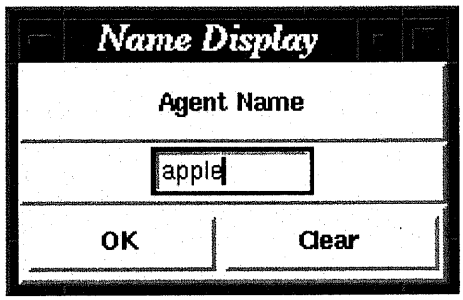


Figure 4.17: Name Display

4.5.4 Tracking Agent の生成

Main Display において Make new T.A ボタンをクリックすると Make Display（図 4.18）が現れる。

次に、Yes ボタンをクリックすると Personal Agent は Server Agent に Tracking Agent 生成を要求し、Server Agent は Tracking Agent を生成する。本実装において、Server Agent は Personal Agent が要求を出した時点での Mobile Agent の存在するホストに Tracking Agent を生成することにしている。

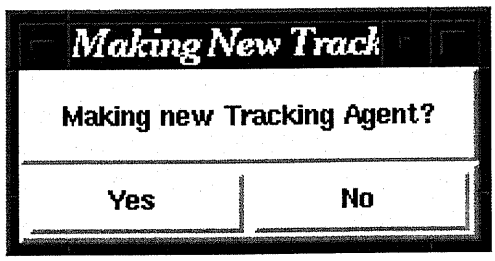


Figure 4.18: Make Display

4.5.5 登録

インターフェイスは次のようになっている。Main Display において Register ボタンをクリックすると、Register Display（図 4.19）が現れる。リストボックスに、現在登録可能な Tracking Agent の名前、トピック、存在するホスト名からなるリスト（以下 Tracking Agent リスト）が表示され、ユーザがいずれかをクリックすると、選択された Tracking Agent の情報が上部のエントリーに表示される。そこで Register ボタンをクリックすれば、Mobile Agent は選択された Tracking Agent に登録される。ただし、登録可能な Tracking Agent が存在しない場合リストボックスにはその旨を示すメッセージが表示され選択不可能となっている。

実装手法は次のようになっている。Main Display において Register ボタンがクリックされると、Personal Agent は Server Agent に Tracking Agent リストを要求し、返送されたリストを得た後、Register Display のリストボックスに示す。その後、Register Display において Register ボタンがクリックされると Mobile Agent に選択された Tracking Agent の名前、存在するホスト名を送信する。それにより、Mobile Agent は指定された Tracking Agent に登録（agent_meet

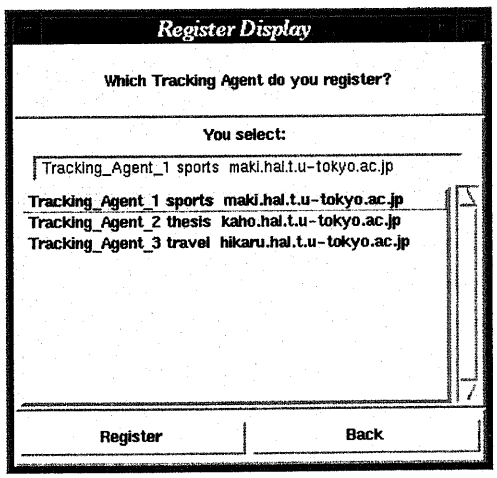


Figure 4.19: Register Display

によりダイレクトコネクションを張ること）要請を出すことになる。

Tracking Agent は、まず生成後一定時間、Mobile Agent の登録要求を待つ。時間内に登録要求がない場合 Tracking Agent は自動消滅する。この際、Tracking Agent は Server Agent に消滅することを知らせ、それに従って Server Agent は Tracking Agent リストからその Tracking Agent に関する情報を削除する。一方、登録要求があった場合、その Mobile Agent とダイレクトコネクションを張り、Mobile Agent の登録 ID、名前、登録時のホスト名、ソケット番号からなる登録リスト（以下 Mobile Agent リスト）を作る。初めの登録以後は、メインループ 1 回毎に登録要請があるかを調べ、あった場合上述した Mobile Agent リストへ Mobile Agent の情報

を追加する。

4.5.6　削除

インターフェイスは次のようになっている。Main Display において Delete ボタンをクリックすると、Delete Display （図 4.20）が現れる。リストボックスに、Mobile Agent が現在登録している Tracking Agent の名前、トピック、存在するホスト名、ソケット番号からなるリスト（以下 Register リスト）のうちソケット番号を除く 3 要素が表示される。

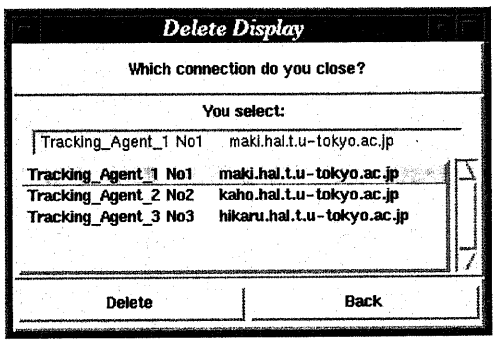


Figure 4.20: Delete Display

実装手法は以下の通りである。Delete Display において Delete ボタンがクリックされると、Personal Agent は Mobile Agent に選択された Tracking Agent のソケット番号を delete イベントとして送信し、Register リストから選択された Tracking Agent に関する情報を削除する。Mobile Agent は受信したソケット番号を持ちいて削除要求を Tracking Agent に送信する。

Tracking Agent は、Mobile Agent リストから削除要求のあった Mobile Agent の情報を削除を行なう。

4.5.7　移動

インターフェイスは次のようになっている。Main Display において Jump ボタンをクリックすると、Jump Display （図 4.21）が現れる。 ここで目的のホスト名を入力しする（研究室内のホスト名の場合は省略可能）。

次に、Request ボタンをクリックすると、現在登録している Tracking Agent に移動要求が送信される。そして Jump Display が図 4.22になり、すべての Tracking Agent からの移動許可がおりると、図 4.23になる。

ここで Jump ボタンをクリックすれば目的のホストへ Mobile Agent は移動し、Cancel ボタンをクリックすれば、移動は中止され、Display は消える。また、図 4.21において、Back ボタンをクリックしても同様に移動は中止され、Display は消える。

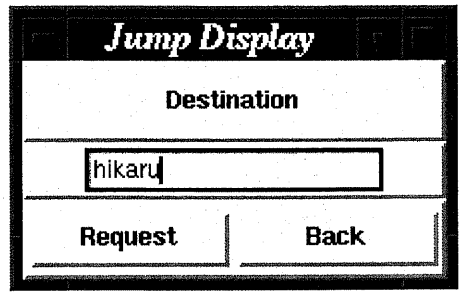


Figure 4.21: Jump Display1

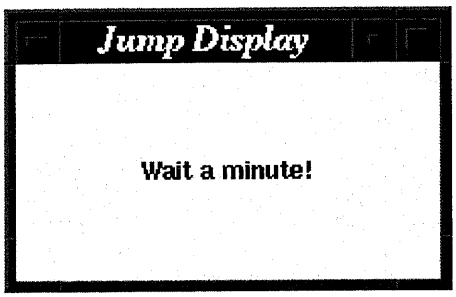


Figure 4.22: Jump Display2

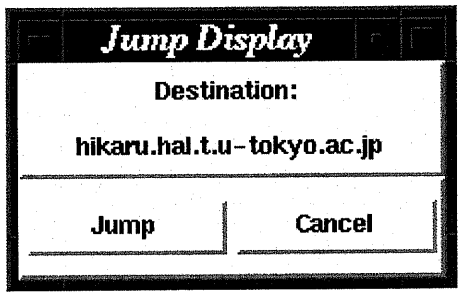


Figure 4.23: Jump Display3

次に、実装手法を図を交えて説明する。

1. まず、Mobile Agent は現在登録している Tracking Agent へそれぞれ移動要求を送信する。
2. Tracking Agent は現在いるホストに Stock Agent を生成し、ダイレクトコネクションを張り、得たソケット番号を Mobile Agent リスト内の Mobile Agent のソケット番号と入れ換える。次に Mobile Agent へ移動許可 (Jump flag) を送信し、Mobile Agent とのダイレクトコネクションを切る。
3. Mobile Agent も Tracking Agent とのダイレクトコネクションを切った後、目的のホストへ移動する。またこの間 Mobile Agent 宛に Tracking Agent に来たメッセージは Stock Agent へ送信される。
4. Mobile Agent は移動後 Tracking Agent へ再接続要求を送信する。Tracking Agent はこの要求を Stock Agent へ送信するとともに Mobile Agent とダイレクトコネクションを張る。また 2 で入れ換えたソケット番号をこの時得たものと再び入れ換える。一方 Stock Agent は蓄えたメッセージの数を送信した後メッセージを Mobile Agent 宛に Tracking Agent へ送信し、消滅する。Tracking Agent は蓄えられていたメッセージを Stock Agent から送信されてきた数に従って、すべて Mobile Agent に送信した後通常の仕事を開始する。

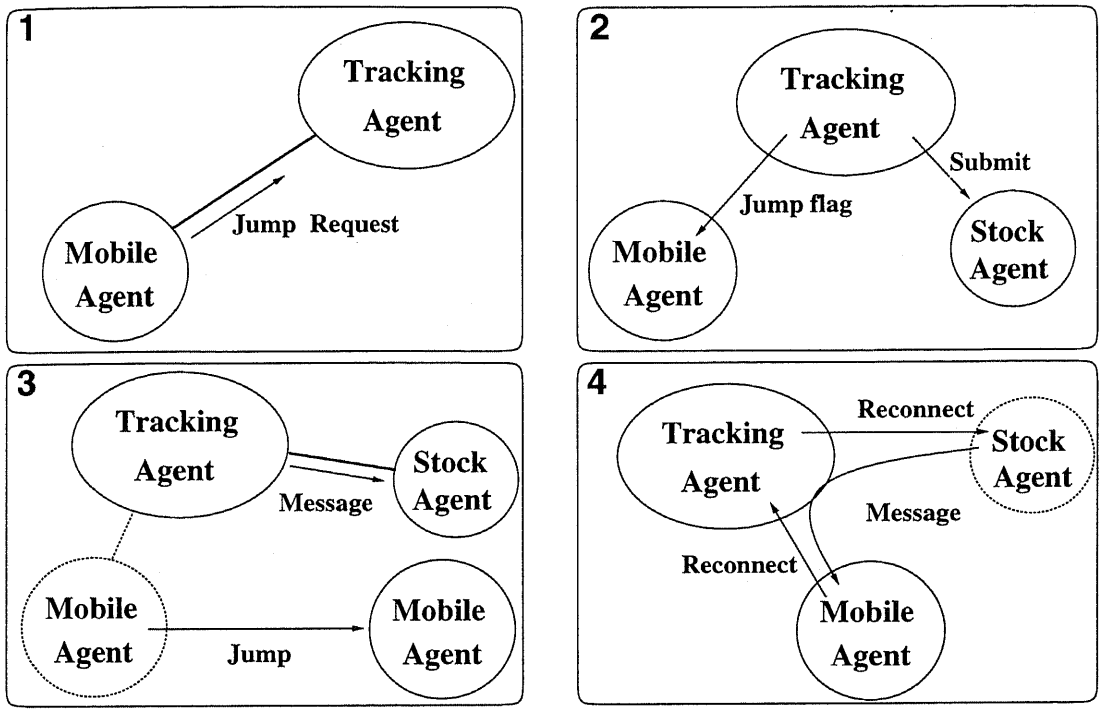


Figure 4.24: 移動手順

4.5.8 メッセージ送受信

まず送信側インターフェイスは次のようになっている。Main Display において Write ボタンをクリックすると、Write Display （図 4.25）が現れる. 相手の Mobile Agent の登録している Tracking Agent とそこでの登録ＩＤを指定し、メッセージを入力した後、Send ボタンをクリックすればメッセージは送信される。Clear ボタンにより入力をすべて消すことが可能であり、Back ボタンをクリックすると Write Display は消える。

次に、受信側インターフェイスについてである。Main Display において Read ボタンをクリックすると、Read Display（図 4.26）が現れる。この時未読のメッセージがあればそのうちもっとも早く届いたものが、無ければ今までの届いたメッセージのうちもっとも新しいものが表示される。また表示されるのはメッセージ番号、送り手の TrackingAgent 、登録ＩＤそしてメッセージである。Next ボタンにより次の、Previous ボタンにより前のメッセージを読むことが可能である。Back ボタンをクリックすると ReadDisplay は消える。

実装手法は以下の通りである。Personal Agent、Mobile Agent 間においては、メッセージ送受信ともにイベントとして処理される。以下に Personal Agent、Mobile Agent、Tracking Agent 間におけるメッセージ送受信の手順を示す。

1. Personal Agent から Mobile Agent へメッセージがイベントとして送信される。

2. Mobile Agent はメッセージを {コード 宛先 メッセージ} という形で、Tracking Agent に送信する。

3. Tracking Agent はコードにより処理を行ない、それがメッセージであった場合 {コード メッセージ 送り主} として目的とされるエージェントに送信する。

4. Mobile Agent は受信した後、コードによりそれがメッセージであった場合、 Personal Agent にメッセージイベントとして 送信する。

5. Personal Agent は受信したメッセージをメッセージ番号とともにメッセージリストに追加、Main Display に未読メッセージ数を表示といったイベント処理を行なう。

手順においても述べたが受信されたメッセージには、メッセージ番号が付けられこの番号に未読か否かの識別子をつけることにより、前述した Read Display におけるメッセージの表示機能を実現した。

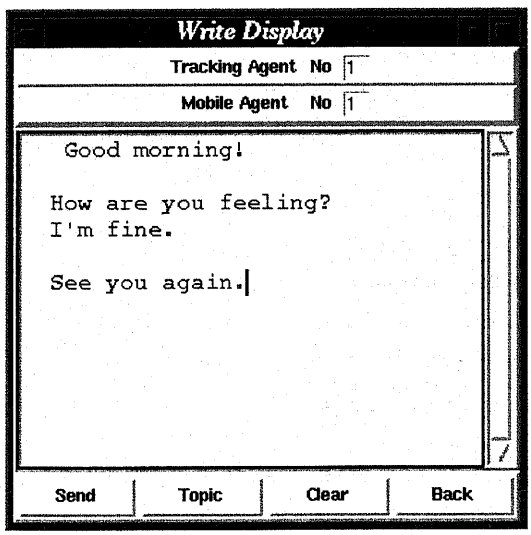


Figure 4.25: Write Display

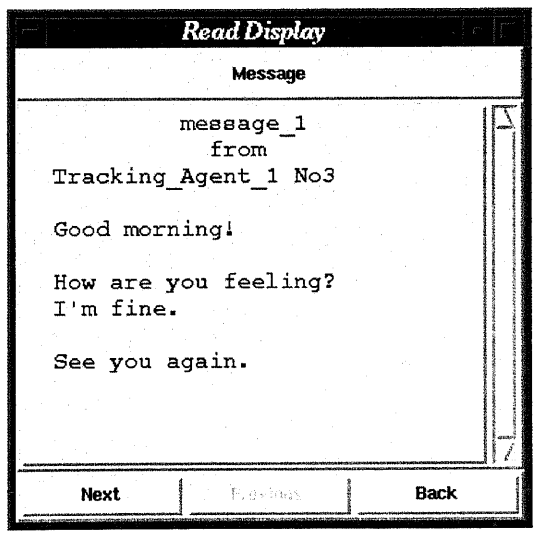


Figure 4.26: Read Display

4.5.9 トピック変更

まずインターフェイスだが、Main Display において Write ボタンをクリックする。次に現れた Write Display において、トピックを変更したい Tracking Agent を指定（図 4.25上部に入力）する。その後、Topic ボタンをクリックすると、Write Display が消えた後、Topic Display（図 4.27）が現れる。Topic Display において、トピックを入力し Set ボタンをクリックすると変更要請が Tracking Agent に送信されトピックが変更される。

入力が NULL 文字列で、Set ボタンがクリックされた場合トピックはディフォルトに設定される。ただし、ディフォルトは NONE である。 Clear ボタンにより入力を消すことが可能であり、

Back ボタンをクリックすると Topic Display は消える。

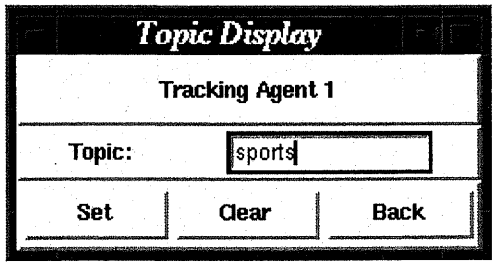


Figure 4.27: Topic Display

4.5.10 コマンド

まずインターフェイスだが、Main Display において Command ボタンをクリックすると、Command Display（図 4.28）が現れる。必要な値またはコマンドを入力し、Value ボタンあるいは Command ボタンをクリックする。しばらく後にコメントとして結果が表示される（4.3.12 参照）。この機能は主に実装時のプログラミングにおいてデバッガとして使用するためのもので

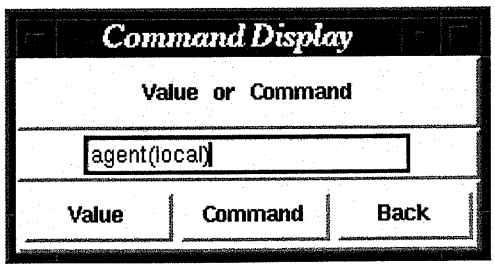


Figure 4.28: Command Display

ある。

4.5.11 登録情報

まずインターフェイスだが、Main Display において Info ボタンをクリックすると、Information Display（図 4.29）が現れる。左上に Tracking Agent リスト、左下に Register リストが表示される。いずれかのリストボックスからリストを選択すると右上にそのリストが、右下にその Tracking Agent の Mobile Agent リストが表示されるようになっている。

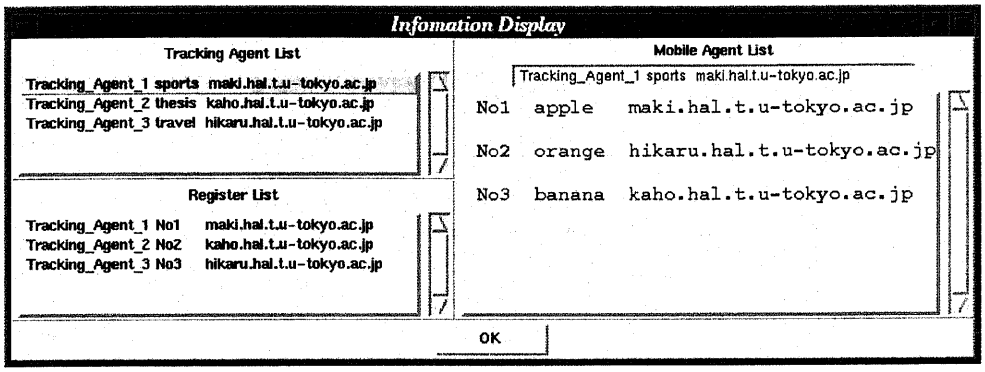


Figure 4.29: Information Display

4.5.12　コメント

様々なイベントがエージェントにより自律的に行なわれる場合、開発者（ユーザ）はその結果をその都度知ることはできない。通信サービスにおいて、ユーザはこのように最終結果のみを知れば良いのだが、実装の段階では、その都度結果を確認することが開発者にとって重要と考えられる。従って本実装ではイベントの結果を、コメントとして Comment Display により表示する

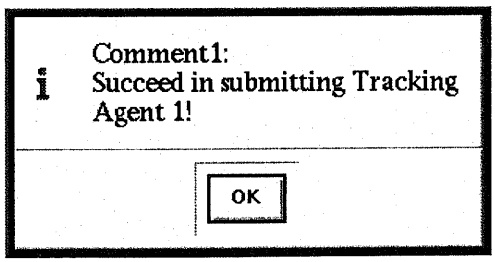


Figure 4.30: Comment Display

ようにした。Comment Display ではコメント番号とコメントの内容が表示される。

例として、Tracking Agent 生成要求を行なった結果通知である Comment Display（図 4.30）を示す。

4.5.13　エラー処理

まず、ユーザの誤入力によるエラーは、エラー番号とエラーメッセージが Error Display により表示される。例として、メッセージ送信において相手の登録ＩＤを誤って入力（存在しない登録ＩＤ: Ｎｏ２を入力）した場合に表示された Error Display （図 4.31）を示す。次に、プログラムにおけるエラー処理として主に以下の３つを実装した。

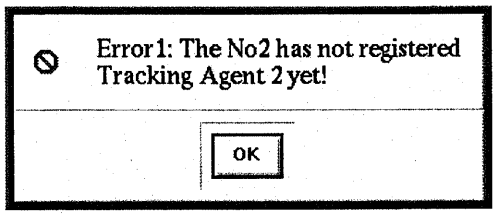


Figure 4.31: Error Display

- エラーが予測されるものは、catch コマンドで正常終了するようにした。
- 親エージェントは、すべての子エージェントを終了させた後、終了する。
- 子エージェントは、親エージェントがエラーにより異常終了した場合、制御不能となってしまうため子エージェントは一定時間毎に親の存在を確認しその存在が確認できない場合スクリプトを自動的に終了するようにした。
- メッセージ送信は事前に受け手が存在するか確認した後、送信するようにした。これは、受け手が存在しない場合でも受け手のホストに名前のないエージェントがいた場合エラー処理されず正常にメッセージが送信されてしまうためである。

4.5.14 実装結果

実装された通信体系において、2つの Mobile Agent 間の Tracking Agent を介したメッセージ送受信（図 4.32）において、4.3.3と同様な処理時間測定を 4.5.10にて説明したコマンドウィンドウを行なった結果、47msec という平均処理時間を得た。

これは、コネクションを張った2つの Mobile Agent 間のメッセージ送受信に比べて約 9 倍に当たる。主な原因は、単純に見て 3 つのエージェント間のメッセージ送受信になっているためと、Tracking Agent においてメッセージがコードにより分類処理されるためである。

それでも、agent_send、agent_event によるコネクションを張らないエージェント間通信に比べると処理時間は約 3/4 で済むという結果が得られた。

4.6 今後の展望

今回の実装において、ネットワーク上を移動可能な Mobile Agent と比べ、Tracking Agent は生成時のホストに固定であった。しかし、登録している Mobile Agent が移動することにより両者の通信距離は増大することが容易に予測される。従って、Tracking Agent は登録しているすべての Mobile Agent と最も効率的にメッセージの送受信ができる位置に移動する必要がある。これは、両者間の通信時間に閾値を設定しそれに従って Tracking Agent が移動を行なうようにす

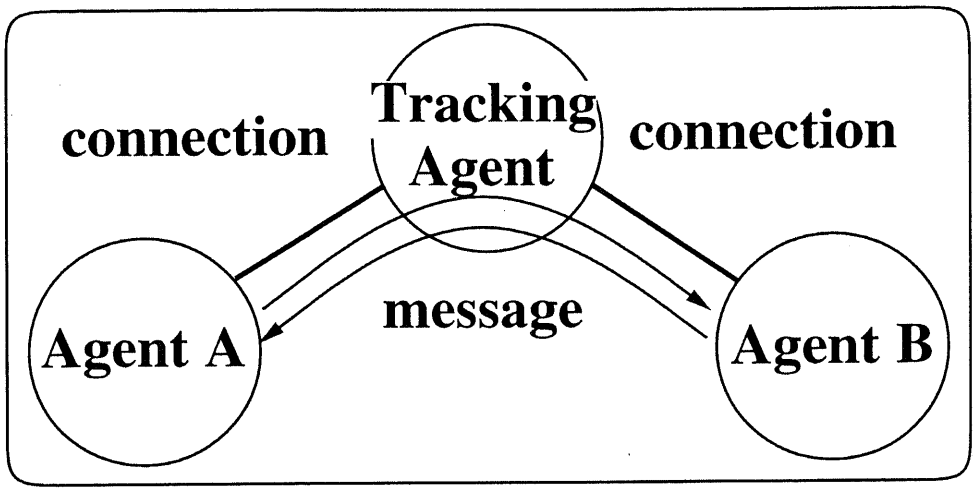


Figure 4.32: エージェント間通信（実装後）

Table 4.3: 平均処理時間の比較

	コマンド	Connection	平均処理時間
実装前	agent_send,agent_event	無し	60msec
実装前	tcpip_write	有り	5msec
実装後	tcpip_write	有り	47msec

ればよい。移動方法は、本実装における Mobile Agent と同様な方法が考えられるが、登録している Mobile Agent が多数の場合、処理にかなりの時間が必要とされることが予測されるため、あまり好ましい方法とは言えないと言える。

従って、Tracking Agent の移動決定アルゴリズムや移動手順は今後検討が必要である。

また、セキュリティも重要な課題である。登録要求を行なう Mobile Agent を Tracking Agent が制限したりする認証機構も必要になる。これは Mobile Agent にネットワーク上での ID を携帯させ、すべてのエージェント間通信においてこの ID が必要な環境を整えればよいであろう。

Bibliography

[1] AgenTalk: マルチエージェント協調プロトコル記述. http://www.cslab.tas.ntt.jp/at/at_home_j.html.

[2] Aglets Software Development Kit. http://www.trl.ibm.co.jp/aglets/.

[3] April World. http://www.fujitsu.co.jp/hypertext/Products/Software/April/.

[4] D'Agents. http://www.cs.dartmouth.edu/~agent/.

[5] Jacl and Tcl Blend. http://sunscript.sun.com/java/.

[6] Knowledge Interchange Format (KIF). http://logic.stanford.edu/kif/.

[7] Object Space Voyager. http://www.objectspace.com/developers/voyager/.

[8] Odyssey. http://www.genmagic.com/technology/odyssey.html.

[9] OMG Object Management Group. http://www.omg.org/.

[10] SunScript. http://sunscript.sun.com/.

[11] TACOMA Operating system support for networking agents. http://www.cs.uit.no/DOS/Tacoma/.

[12] UMBC KQML Web. http://retriever.cs.umbc.edu/kqml/.

[13] Robert S. Gray. *Agent Tcl: Alpha Release 1.1.* Department of Computer Science, Dartmouth College, Dec 1995.

[14] Rob Leathern. AgentTcl. http://www.cs.dartmouth.edu/~agent/.

[15] Ontolingua /Stanford / NICI Nijmegen. http://ontolingua.nici.kun.nl/.

[16] 西ヶ谷岳, 飯田一朗. エージェント指向のパーソナル通信網:duet. 信学技報, Vol. CS94-64. pp. 1–6, Aug 1994.

[17] 西ヶ谷岳, 栗田敏彦, 飯田一郎, 村上孝三. エージェント指向ネットワーキングアーキテクチャ DUET の提案. 電子情報通信学会論文誌, pp. 216–225, 1996.

[18] 五十嵐弓将, 野原龍男, 石井啓之. パーソナル通信サービスにおけるエージェント間調整方式の検討. 信学技報, Vol. CS96-6, , Apr 1996.

[19] 竹田憲司, 谷英明, 西田竹志. パーソナル通信サービスに向けたユーザエージェントの構成. Technical Report CS94-217, 信学技報, 1995.

Chapter 5

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

本章では一対多のエージェント通信においてエージェントを階層的に配置し、さらに動的にその構造を変化させることによってエージェント間通信に伴うトラヒック及び中心的なエージェントの負荷を軽減することを提案している。

5.1 マルチエージェントシステム

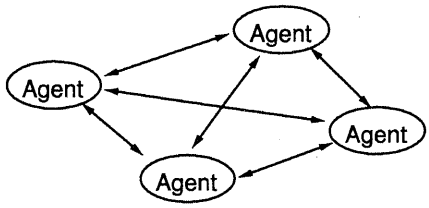
ネットワーク内に存在し、ユーザーの代理となって自律的にはたらくプロセスを総称して、ネットワークエージェント、中でもネットワーク上を移動しながらリモートで仕事を行なうモバイルエージェントと呼び、様々な研究が行われている [6,5,4,2,1]。また、この中で特に複数のエージェントを協調させ、目的を達成させるものをマルチエージェントシステムという。マルチエージェントシステムについていろいろなシステムが考案されており [12,7,9,11,13,8]、これらは大きく次の2つにわけられる。すなわち、

- 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

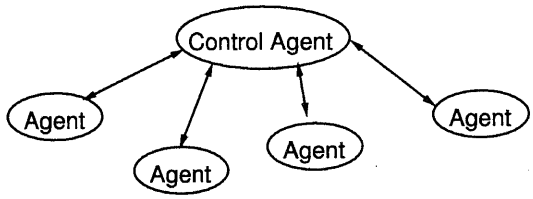
である（図 5.1）。

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



自律分散型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

Figure 5.1: マルチエージェントシステム

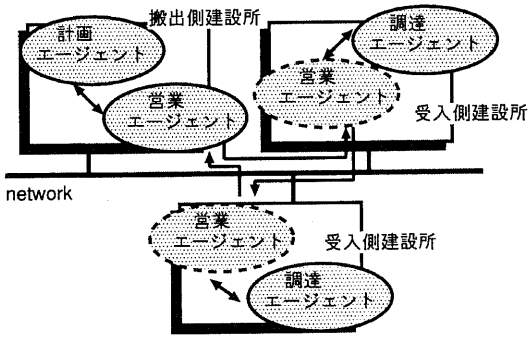


Figure 5.2: リサイクル調整支援へのマルチエージェント技術の適用

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 5.2）。

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ほうが便利な場合がある。例えば、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一カ所に集め、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、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 [8] 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。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な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。但し、実際使用される時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。

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 [10] 。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 5.7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。

以上、5.2章では階層的マルチエージェントとその利点を論じ、5.3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。5.4章では実装について、5.5章、5.6章にてその実装による実験をまとめる。5.7章にて本提案のシステムの応用に触れ、5.8章でまとめる。

5.2　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 1 対多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 は一つの 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 5.4)。

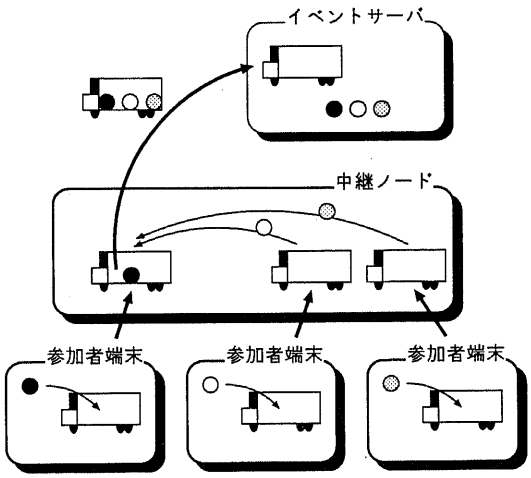


Figure 5.3: 移動エージェントを用いたデータ集約システム

そこで本論文では、図 5.5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わけ、その一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。

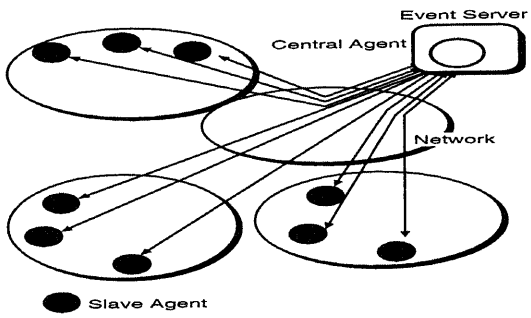


Figure 5.4: 集中制御型マルチエージェントシステム

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 及び 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

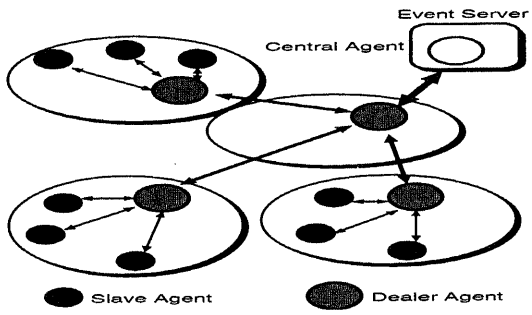


Figure 5.5: 階層的マルチエージェントシステム

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挙げられる。

- 負荷分散

  負荷の集中は、 Central Agent での処理が一つのホストで行な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。例えば 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。もしある一つの 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度に行な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。

5.3 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、もう一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。

5.3.1　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 で集積してひと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、ヘッダのオーバー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。

5.3.2　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 を除く全てのエージェントは必ず一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に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良い。このため処理中のシステムを止めることなく常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、及び移動の詳細については 5.4.1,5.4.2節で述べる。

5.3.3　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なう場合、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、

$$D \simeq D_\beta \tag{5.1}$$

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、

$$T_{RTT} = D_{\beta}^{SC} + D_{\beta}^{CS} \tag{5.2}$$

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

$$T_{interval}^{C} = \frac{T_{RTT} + T_{rest}}{n} \tag{5.3}$$

$$T_{interval}^{S} = T_{RTT} + T_{rest} \tag{5.4}$$

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

$$T_{RTT} > nT_{send}^{CS} - T_{rest} \tag{5.5}$$

$$T_{RTT} > T_{send}^{SC} - T_{rest} \tag{5.6}$$

である必要がある。ここで、$T_{send}^{CS} > T_{send}^{SC}$であることより、(5.5) 式があるので (5.6) 式は無視してよい。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 (5.3) は次のように修正される。

$$T_{interval}^{C} = T_{send}^{CS} = \frac{T_{RTT} + T_{rest} + T_{wait}^{C}}{n} \tag{5.7}$$

このとき、ラウンドトリップタイム T'_{RTT} は

$$T'_{RTT} = T_{RTT} + T_{wait}^{C} = nT_{send}^{CS} - T_{rest} \tag{5.8}$$

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なった場合について考える。このとき T_{RTT} は、

$$T_{RTT} = D_{\beta}^{SD} + T_{collect}^{D} + D_{\beta}^{DC} + D_{\beta}^{CD} + D_{\beta}^{DS} \tag{5.9}$$

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、

$$T^{C}_{interval} = T_{RTT} + T_{rest} \tag{5.10}$$
$$T^{D}_{interval} = \frac{T_{RTT} + T_{rest}}{1+m} \tag{5.11}$$
$$T^{S}_{interval} = T_{RTT} + T_{rest} \tag{5.12}$$

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 $T^{C}_{interval} > T^{CD}_{send}$, $T^{D}_{interval} > T^{DC}_{send}$, $T^{D}_{interval} > T^{DS}_{send}$, $T^{S}_{interval} > T^{SD}_{send}$ である。ここで $T^{C}_{interval} = T^{S}_{interval} > T^{D}_{interval}$, であり、Dealer Agent は Slave Agent に近いところに配置するので $T^{DC}_{send} > T^{DS}_{send}$ である。したがって、このシステムが待ち時間が発生しない条件は (5.11) 式から

$$\frac{T_{RTT} + T_{rest}}{1+m} > T^{DC}_{send} \tag{5.13}$$

となる。

ここで $m = n$ とすると $T^{CD}_{send} > T^{DC}_{send}$, $T^{CS}_{send} > T^{DS}_{send}$, $T^{CD}_{send} \simeq T^{CS}_{send} - T^{DS}_{send}$であるから、(5.5)(5.13) 式より

$$\begin{aligned} & nT^{CS}_{send} - (m+1)T^{DC}_{send} \\ > \; & nT^{CS}_{send} - (n+1)T^{CD}_{send} \\ \simeq \; & (n-1)\left(T^{CS}_{send} - T^{DS}_{send}\right) \\ > \; & 0 \end{aligned} \tag{5.14}$$

したがって、(5.5) 式が成り立たなくても (5.13) 式が成り立つ状態が存在する。すなわち、Dealer Agent が無くて待ち時間が発生する場合に (5.13) 式が成り立つ場合に 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。特に全ての 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 (5.13) 式の成り立つ範囲が広くなる。

以上より、(5.5) 式が成り立つ場合に (5.13) 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関わらず (5.5)(5.13) 式を満たさない場合には Dealer Agent がいない方が良いので、この Dealer Agent は自分の全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。

5.4　階層的マルチエージェントシステムの実装

本節では前節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。

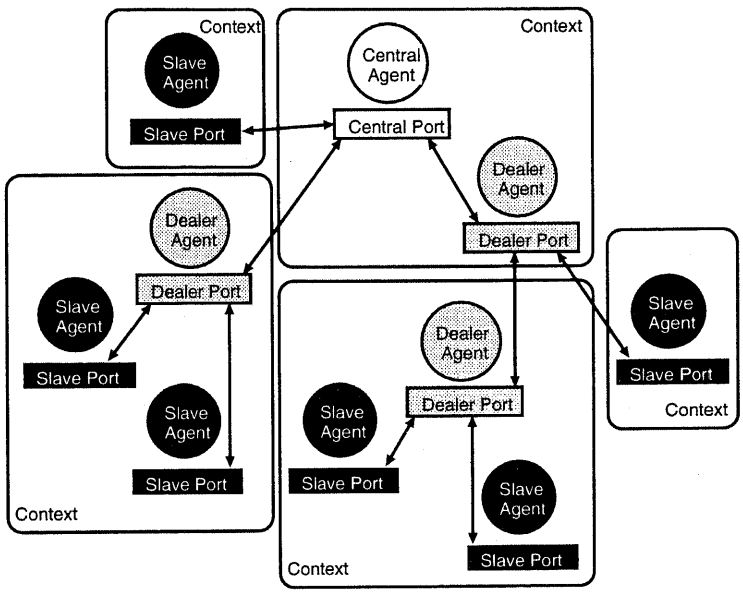


Figure 5.6: 実装した階層的マルチエージェントシステムのモデル

図 5.6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 と通信を行な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)。

5.4.1　エージェント管理

5.3.2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 (以下 Master Agent と呼ぶ) の Proxy と、直接下位の Dealer Agent または Slave Agent (以下 Sub Agent と呼ぶ) 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 5.7)。

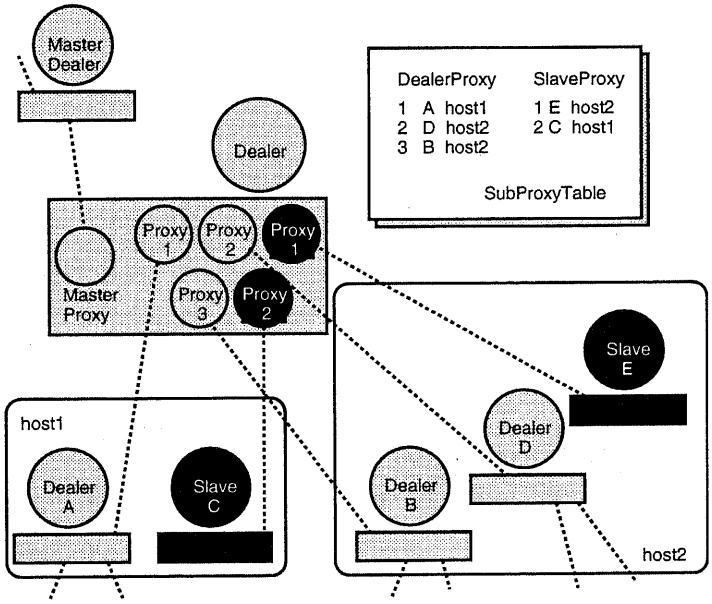


Figure 5.7: エージェント管理

5.4.2　エージェントの名前づけ

エージェントは、新たな Sub Agent を生成する際、及び新たな Agent が移動してきた際に、名前が一意になるように名前付けを割り当てる。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 とする (図 5.8)。

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なう。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。

5.4.3　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 5.9に示す。

- まず移動の意志を持つエージェント (以下 Move Agent と表わす) は、自分に接続する全てのエージェント (Master Agent と全ての 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 (以下 Connected Agent と表わす) は、自分が移動のオペレーションの最中であったりなどの都合の悪い場合を除いて、了解の返事を送る。

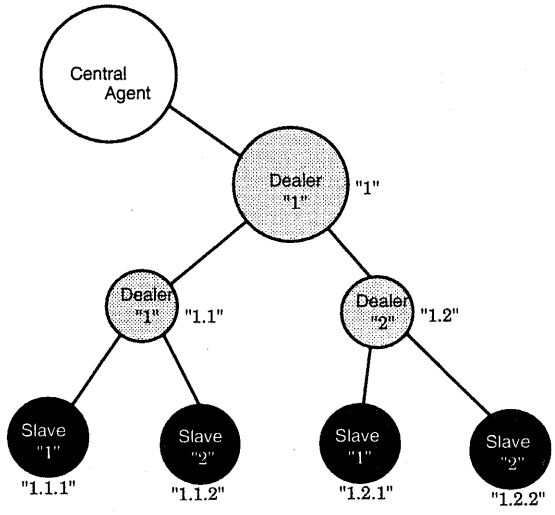


Figure 5.8: 名前管理

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)。

- 全ての 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 は全ての 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。

5.4.4　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 5.10、図 5.11、図 5.12 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

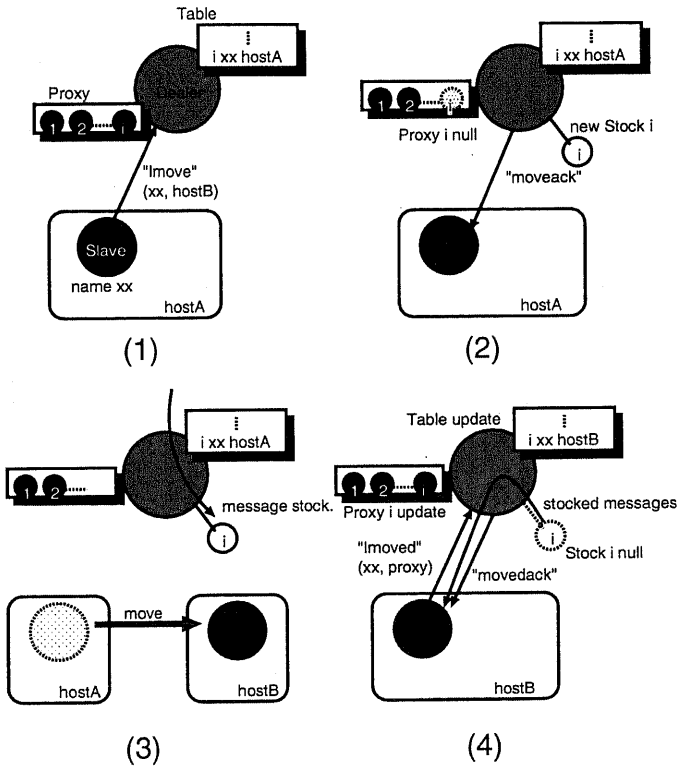


Figure 5.9: エージェントの移動

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 5.10 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 5.11 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 は自分の全ての 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なわれる。

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。全ての Sub Agent か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 は全ての 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 5.11 右)。

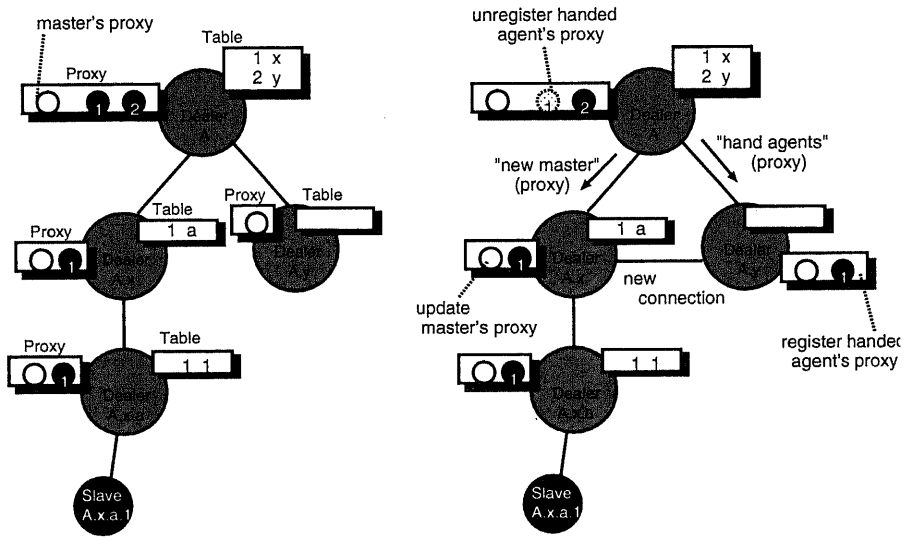


Figure 5.10: Sub Agent を Sub Agent に渡す変化 1

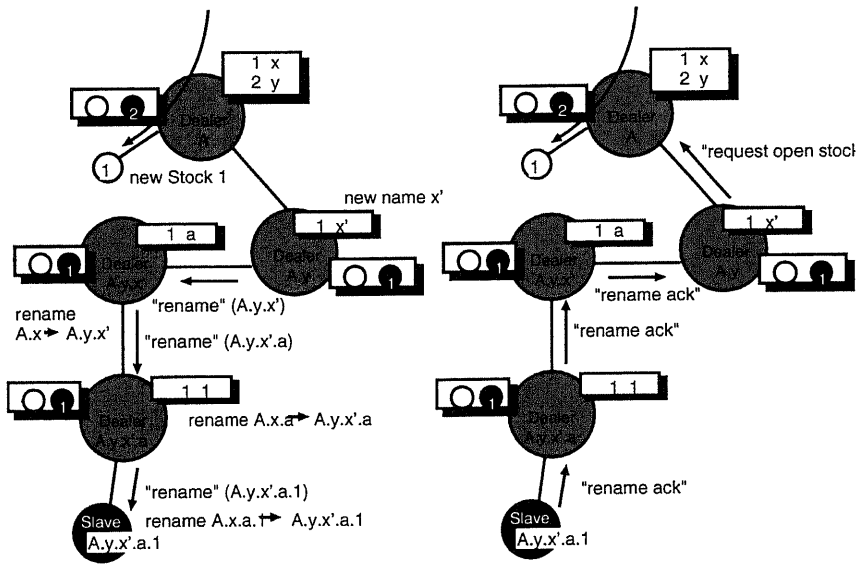


Figure 5.11: Sub Agent を Sub Agent に渡す変化 2

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。全て渡し終えたら Stock を消滅させ、最後に SubProxyTable から Mover を抹消する (図 5.12)。

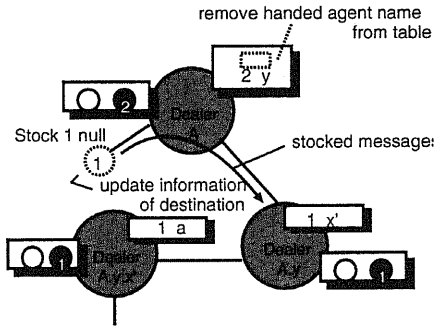


Figure 5.12: Sub Agent を Sub Agent に渡す変化 3

5.4.5　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) がき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、5.3.3 節の式 (5.5) が成り立つか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 $T_{thr} = T^{CA}_{relay} + \alpha$ (α はシステムの安定性を保つため必要) より低くなり、式 (5.5) が偽になると (5.13) 式も考慮した上で 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} をある程度越えた場合 (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

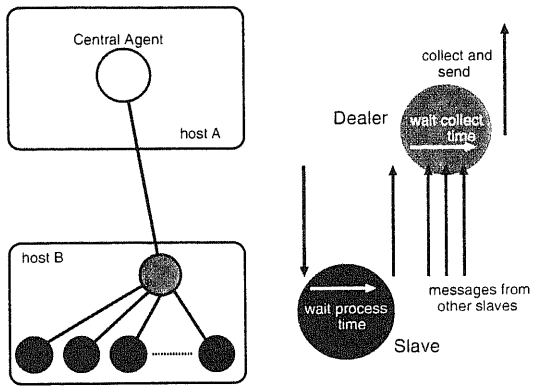


Figure 5.13: 実験の概略

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。

5.5 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なった。

5.5.1 実験方法

図 5.13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な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 5.13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なった。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 [3] を用いた。

5.5.2 実験結果

前節で説明した実験の結果を図 5.14(a)から図 5.15(f)までに示す。図 5.14(a)、図 5.15(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 5.14(b)、図 5.15(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 $10, 50, 100, 500msec.$ 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 $0, 10, 50, 100, 500msec.$ の 5 通りについて計測した。

5.5.3 考察

Dealer Agent がいない場合についてまず考察する。図 5.14(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 $40msec.$ であり、また、図 5.15(a)より待ち時間が発生したときの $T_{interval}^{C} (= T_{send}^{CS})$ は約 $60msec.$ であることがわかる。

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 5.15(a)において式 (5.5) 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 $0msec.$ の場合は Slave Agent の数が 2 つ以上で既に該当するのに対し、$T_{process}$ が $500msec.$ 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わかる。

輻輳している状態で、図 5.14(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 (5.8) から

$$T_{RTT} = T_{send}^{CS} n - T_{rest} \tag{5.15}$$

であるので、$T_{send}^{CS} = 55msec.$ 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

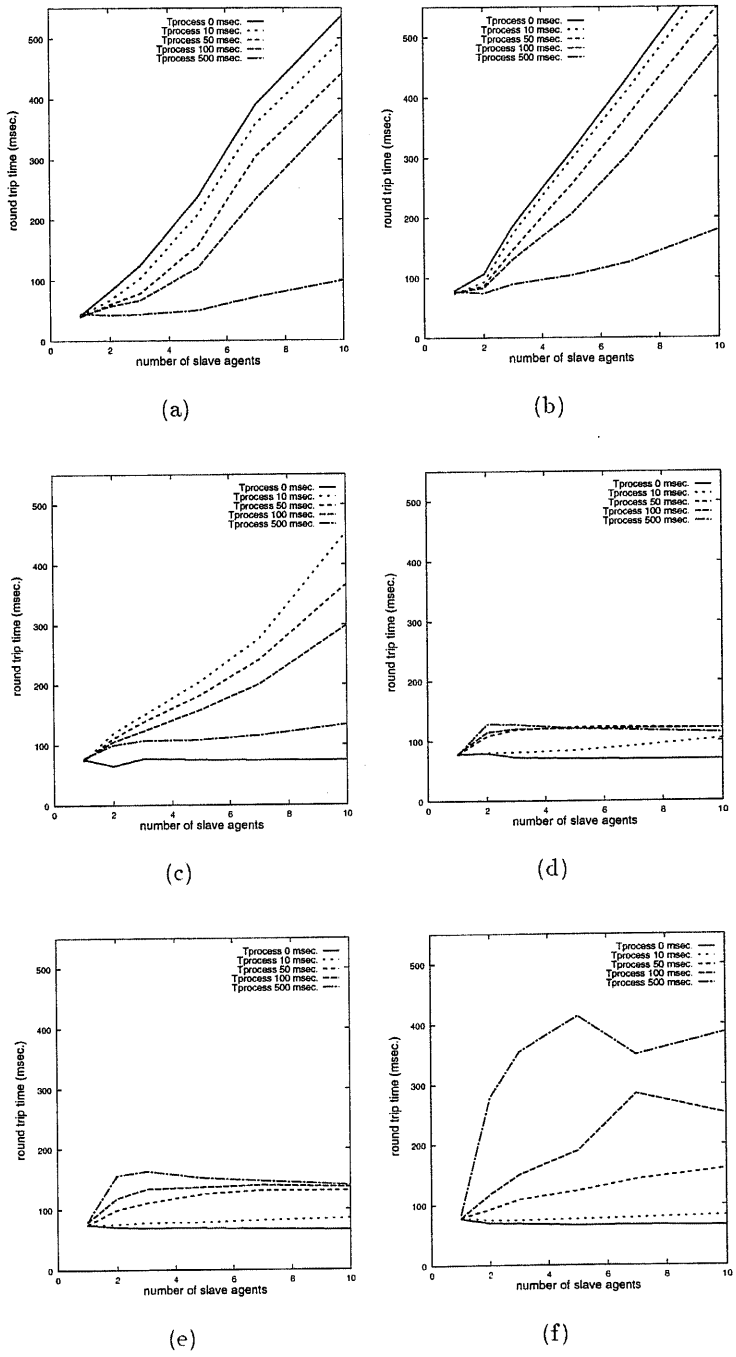


Figure 5.14: Slave Agent と Central Agent 間のメッセージの Round Trip Time. (a): Dealer Agent 無し, (b): Dealer Agent 有り且つ蓄積時間=0, (c),(d),(e),(f),: Dealer Agent 有り且つ蓄積待ち時間最大 10, 50, 100, 500ms.

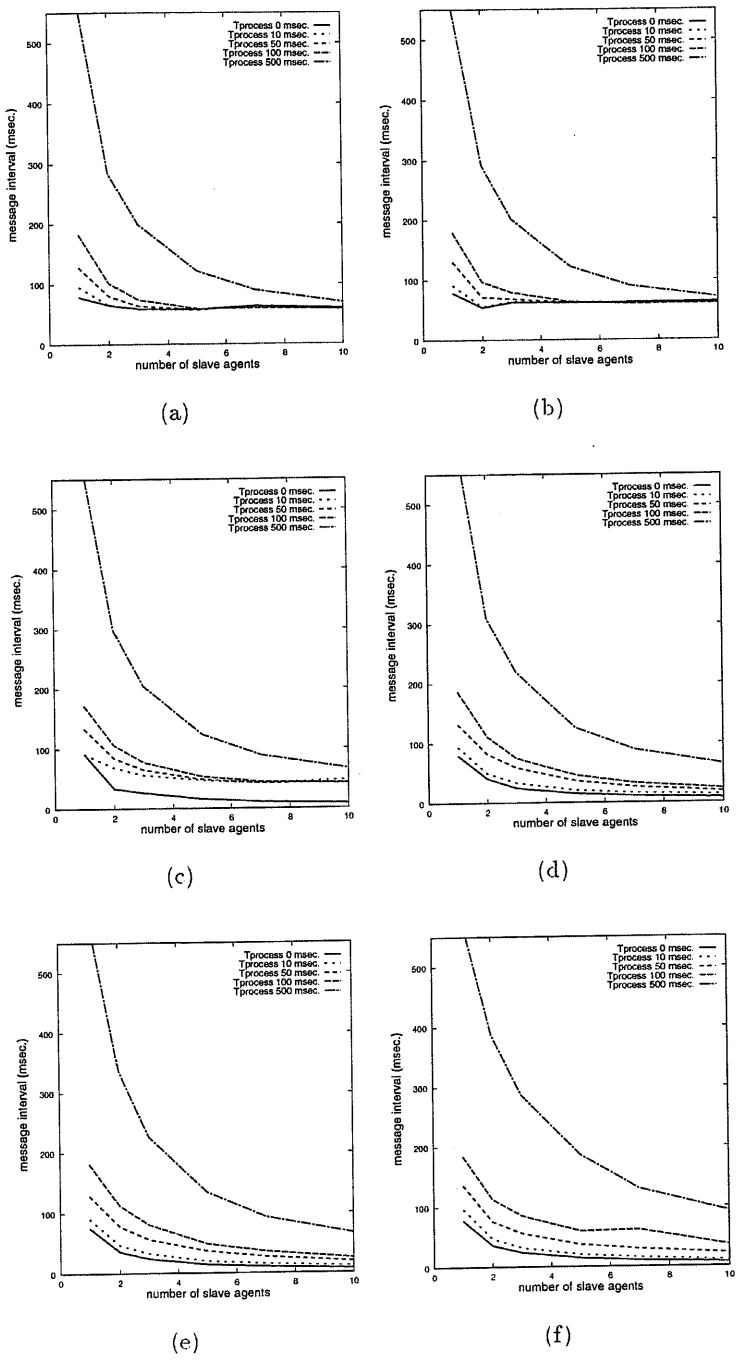


Figure 5.15: Central Agent における平均メッセージ受信間隔. (a): Dealer Agent 無し, (b): Dealer Agent 有り且つ蓄積時間=0, (c),(d),(e),(f),: Dealer Agent 有り且つ蓄積待ち時間最大 10, 50, 100, 500ms.

$$T_{rest} = 0.94 \times T_{Process} + 48.1 \tag{5.16}$$

の関係になっているので、おおよそ $50msec.$ のバイアスがかかっていると考察できる。これは式 (5.3) 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。

以上を踏まえて、次は Dealer Agent がいるときの結果を考察する。まとめ送りを行なわない場合 (図 5.14(b)、図 5.15(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 5.14(a) と 図 5.14(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 5.14(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 $75msec.$ と読みとれるので、$D_\beta^{SD} + D_\beta^{DS}$ が約 $35msec.$ であることがわかる。一方メッセージ処理時間 T_{send} は図 5.15(b) から約 $60msec.$ 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、同ホスト同士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わかる。

輻輳している状態で、図 5.14(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。

$$\begin{array}{ll} T_{process} = 0 & T_{RTT} = 64.0n - 8.2 \\ T_{process} = 10 & T_{RTT} = 61.7n - 12.1 \\ T_{process} = 50 & T_{RTT} = 59.5n - 43.4 \\ T_{process} = 100 & T_{RTT} = 56.5n - 80.5 \end{array} \tag{5.17}$$

係数はほぼかわらないので、ここでも同ホスト同士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わかる。また、

$$T_{rest} = 0.74 \times T_{Process} + 6.5 \tag{5.18}$$

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

続いてまとめ送りを行なう場合について考察する。まとめ送りを行なう場合は、 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 5.14(b), 図 5.14(c)) は、集積が不十分なためメッセージ数の増加が抑え切れず、輻輳をおこしている。ここで一番トラヒックの多いはずの $T_{process} = 0msec.$ 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 5.14(d)図 5.14(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 5.14(f), 図 5.15(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

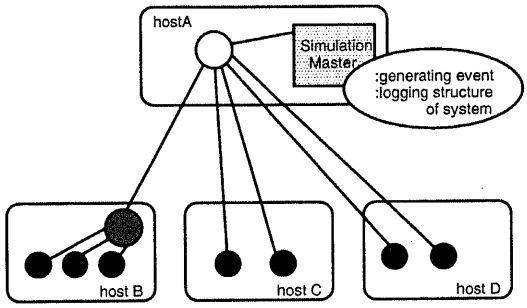


Figure 5.16: 実験の概略

できる。

5.6 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なった。

5.6.1 実験方法

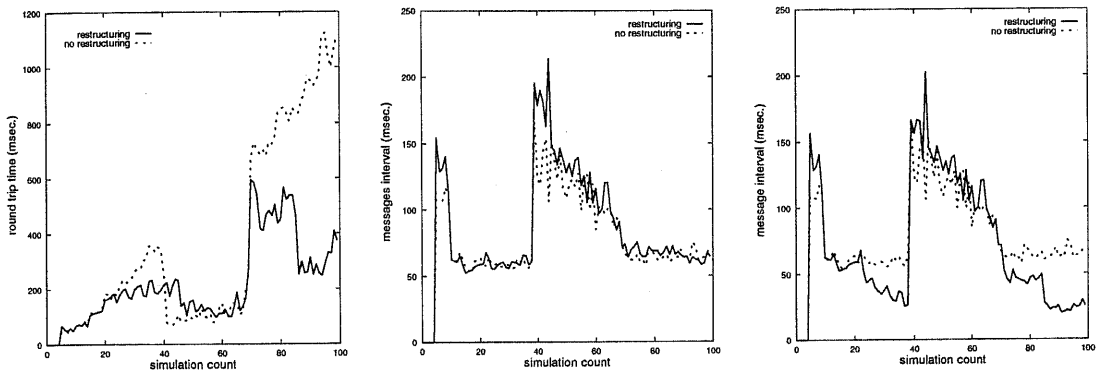
図 5.16 に実験の概略を示す。Central Agent のいるホストに新たに Simulation Master というエージェントを実装し、この Simulation Master がシステムに対し決まったタイミングでイベントを起こしたときにどのようにシステムが動的再構成を行なうかを記録した。イベントは 4 秒毎にカウントされるカウンタによって駆動し、Simulation Master から Central Agent を通じてシステムの各エージェントに通達される。

イベントは、次のように定めた。カウントが進むにしたがって 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。更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 $180ms, 400ms$、host C では $160ms, 300ms$、host D では $75ms, 150ms$ と定めた。

5.6.2 実験結果

実験結果を図 5.17(a) から図 5.18(c) までに示す。図 5.17(a) から 図 5.17(c) までは再構成機能があった場合と、再構成無しで Dealer Agent を用いなかった場合において、ラウンドトリップ



(a) (b) (c)

Figure 5.17: 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.

タイム、メッセージ群受信間隔、メッセージ受信間隔を示している。図 5.18(a) から 図 5.18(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。メッセージ群受信間隔は、 Slave Agent から送られてくるメッセージは必ず一つずつなのでメッセージ群受信間隔を同一となる。

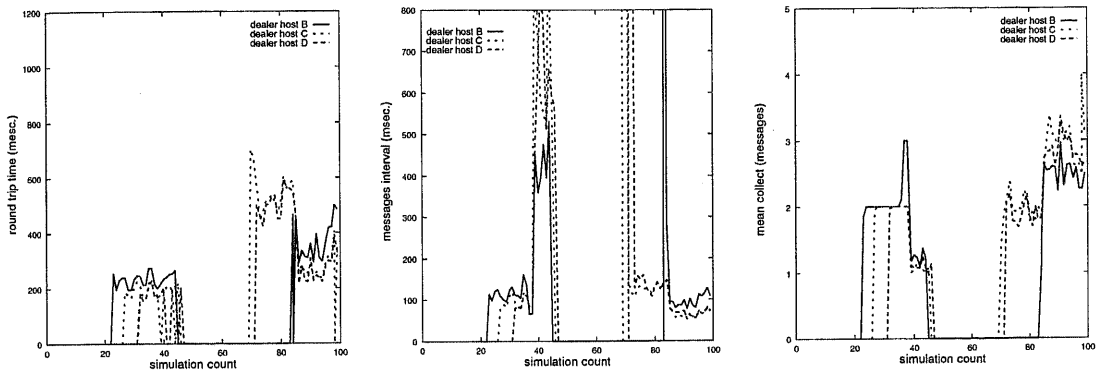
図 5.19 はイベントとシステムの変化の概略をまとめたものである。"s" が打たれている時刻にそのホストに新たな Slave Agent が生成されたことを示し、 "D" の箇所で再構成が行なわれてそのホストに Dealer Agent が送り込まれている。この際、 Central Agent に所属していてそのホストにいる Slave Agent は全てこの新しい Dealer Agent に手渡される。

また、simulation count が 40 及び 69 のところで全ての Slave Agent の $T_{process}$ が切り替わっていることを表す。これに伴って Dealer Agent が消滅し、自分の Sub Agent を全て Central Agent に渡した時刻を "d" で表している。

5.6.3 考察

図 5.17(a) より、システムが状況の変化に即して再構成を行なった結果、ラウンドトリップタイム が向上していることがわかる。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 5.18(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 5.17(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



(a) (b) (c)

Figure 5.18: 再構成ありで 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.

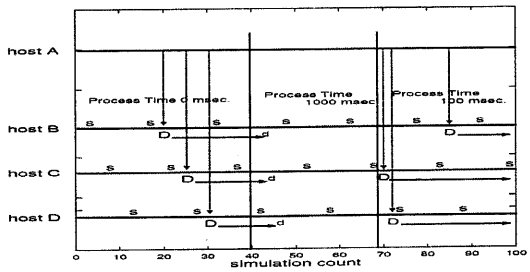


Figure 5.19: システムの変化

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 5.17(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 5.18(c)) メッセージ受信間隔が低下し (図 5.17(c))、それにも関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 5.17(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。

5.7 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、増減する場合

このため、例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取りを行い、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全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。もう一つ、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。

5.8 むすび

本研究ではモバイルエージェントの応用例の一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、全てに自律的な機能をも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。

Bibliography

[1] A. H. Bond and L Gasser. Readings in Distributed Artificial Intelligence. Morgan Kaufmann Publishers, 1988.

[2] Horizon Systems Laboratory. Mobile Agents White Paper. `http://www.hri.com/HSL/Projects/Concordia/MobileAgentsWhitePaper.html`.

[3] 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`.

[4] Pattie Maes. Agents that reduce work and information overload. *Communications of the ACM*, Vol. 37, No. 7, pp. 30–40, Jul 1994.

[5] E. Pitoura and B. Bhargava. Building Information Systems for Mobile Environments. In *the 3rd International Conference on Information and Knowledge Management*, 1994.

[6] Jan Vitec and Christian Tschudin Eds. Mobile Object Systems: Towards the Programmable Internet. In *Second International Workshop, MOS'96*, Jul. 1996. `http://www.icsi.berkeley.edu/~tschudin/lncs-1222.html`.

[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] 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-. 情報処理学会第 56 回全国大会 (平成 10 年前期) 講演論文集, 1998.

[9] 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.

[10] 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, Feb 1999.

[11] 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.

[12] 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.

[13] 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.

Chapter 6

Web とエージェント — 分散データベース —

現在 WEB は情報の宝庫である。しかし、情報が氾濫しているだけで体系的有機的に構成されているわけではないので、求める情報を探し出すのは大変困難である。この氾濫している情報を結び付けて探し出すことを容易にするのが検索システムである。ネットワークに散乱するデータを結び付けてデータベースとするのである。

WEB ブラウジングによるトラヒックを軽減するために Proxy Cache が広く用いられるようになってきている。そこで proxy cache を検索システムに利用し、さらに複数の検索システムを連携する際にモバイルエージェントを利用することを検討してる。その予備的な検討として、本章ではまず分散データベースについて概観し、続いて WEB Proxy Cache へのトラヒックの解析・検討について述べる。

6.1 データベース

6.2 はじめに

データベースシステムはコンピュータがこの世に現われて以来、コンピュータ上の主要なアプリケーションの一つとなっている。近年のハードウェア及びソフトウェア技術の著しい発展により、一台の大型コンピュータが一つの機関に設置されていた時代から、一人一台のワークステーション時代の今日へと大きく変遷して来た。それに伴い、データベース管理システムの応用分野は拡大の一途を辿り、人間の管理能力を超えた大量のデータを効率良く管理、運用するデータ処理機械としての役割はますます重要となってきている。

データベースシステムに大きな変革をもたらした、リレーショナルデータベースモデルは、1970年に IBM サンホゼ研究所の Codd[11] によって提案された。このモデルはネットワークデータモデルや階層データモデルと言った従来からのデータモデルと異なり、データベースを数学的に定

義される relation の集合として定義した。1980 年代に入ると大型汎用機稼働を前提とした商用の RDBMS [1] が数多く登場し、ビジネスデータ管理に RDBMS を利用する例が増えてきた。

一方コンピュータの小型、高性能化、またダウンサイジングの動きに伴い、RDBMS を取り巻く環境も大きく変更され、小型のワークステーションでも RDBMS が十分機能するようになってきた。また、従来の集中管理型システムが崩れ、小型のコンピュータの分散配置が当然となるにしたがって、データベースも集中型から分散型へと変化してきた。

今日では、大規模ネットワーク上に分散されて構築されたデータベースから、個々に蓄積・管理・運用されてきたデータベースをいかに有機的に結合するかまで、分散データベースが扱うべき問題は非常に幅広いものとなってきている。

近年 World Wide Web から情報を取得することが多くなってきたが、WWW サービスはサーバへの負荷の集中とトラヒックの増加を引き起こし、これが問題となってきている。そこで WWW トラヒック・サーバ負荷の軽減とレスポンスの向上を目的とした WWW 用のキャッシング技術のの研究が盛んに行なわれ [9]、特に複数のキャッシュサーバが連係し世界規模のキャッシュツリーの構築が進められている [5]。

本章では分散データベースを堅苦しく考えず、身近にある分散データベースの例として WWW の Cache を取り上げる。6.2.1では分散システムの定義について述べ、6.2.2で分散データベースの基礎を述べる。6.3では WWW 上での分散データベースの例としてプロキシーキャッシュを取り上げる。

6.2.1 分散システム

分散システムを定義付けるには分離と透過という二つのキーワードを用いて次のように表すことができる [28]。

> 分散システムは分離して設置されてコンピュータ間が透過性を保証された形で、協調的に動作するシステム

コンピュータシステムおよびネットワークの小型高性能化、低価格化、及びユーザの要求の多様化による集中型システムの破綻が、分散化を押し進めた原動力である。ダウンサイジングは単にコンピュータの規模を小型にすることではなく、様々なユーザの不定期なジョブをいかに効率良く処理する分散環境を構築するか、という意味であった。ネットワークで接続され自律的に動作するコンピュータ群によって構成される分散システムは、少なくとも従来の集中型コンピュータと同等あるいはそれ以上に柔軟で適用範囲の広い計算能力を提供しながら、かつネットワーク環境において資源を共有できるように設計されていなくてはいけない。さらに、ユーザに対しては単一の統合された環境を利用しているように見える、すなわちシームレスな環境を提供する必要がある。もちろん、個々のコンピュータはそれぞれ物理的に離れた場所に設置されていても問題はなくシームレスな環境が提供されるべきである。

[1] RDBMS(Relational DataBase Management System

身近な分散システムとしては 分散ファイルシステムがある。これはワークステーションの能力と柔軟性を利用し、それらが高速 LAN を介して他のサーバコンピュータ上の資源をアクセスするというもので、クライアント・サーバモデル と呼ばれる。

分散システムの特徴は上にも述べたように分離と透過の二つのキーワードで表すことができる。これらをもう少し詳しく説明する。

- 構成要素の分離
  分散システムの本質的な要件である。ネットワークを介したシステム管理及び統合技術が必要で、基盤技術としては、プロセスの並列実行、構成要素の障害の封じ込め対策、システムを混乱させない障害の回復、セキュリティとロック機構、構成要素の変更機構等が挙げられる。

- 透過性
  ユーザやアプリケーションに、分離を意識させないシームレスな環境を提供することを指す。透過性の要素は次のようなものがある。

  - 位置、移動透過性
    複数のサイトに分散配置されているデータをアクセスする場合に、どこにデータがあるのかをユーザが意識せずにアクセスできる環境を提供することである。
  - 一貫性：アクセス透過性、複製透過性、障害透過性
    複製を複数のサイトに生成することにより、耐障害性は向上する。しかしながら、データの更新や一貫性の維持などは全ての複製されたデータに対して行なう必要がある。このような処理はユーザから隠蔽された状態で実行される。

6.2.2 分散データベース

分散データベースシステム[2]とはコンピュータネットワークを介して論理的に結合された複数のデータベースの集まりとして定義でき [28]、また、分散データベース管理システム[3] は分散データベースシステムを管理し、ユーザに対して分散透過性を保証するようなソフトウェアシステムと定義することができる。ここで、分散透過性とは分散配置されているデータベースをユーザに意識させないことを表す。

分散データベースの定義においては「ネットワークを介して」「論理的に統合されている」という二つのキーワードが含まれている。しかしながら、図 6.1のようなデータベースが一つのノードにあるようなシステムは、従来の集中型システムと何も変わらない。図 6.2のように複数の分離したサイトにデータベースがおかれ、利用者はデータの所在を気にせずアクセスできるものが分散データベースである。

分散データベースとは何が分散されているのかは、以下のように考えることができる。

[2]Distributed DataBase System, DDBS
[3]Distributed DataBase Management System, DDBMS

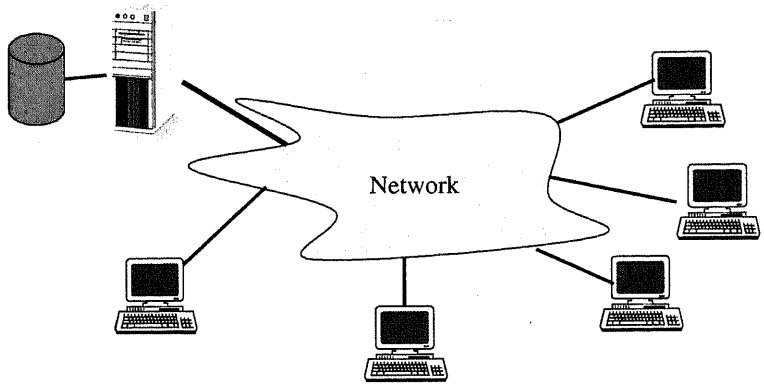


Figure 6.1: ネットワーク統合された集中型データベース

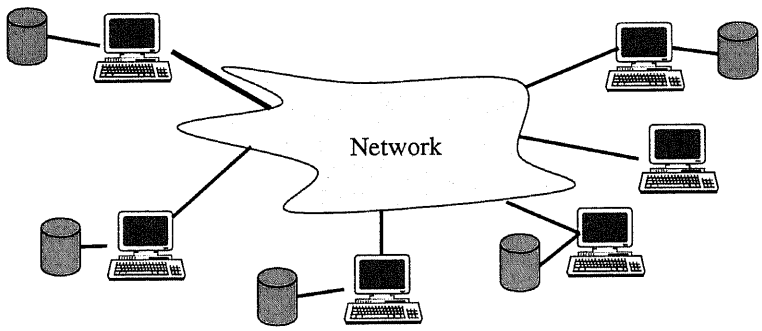


Figure 6.2: 分散データベースの典型的な形態

- データ
  それぞれのノード (サイト) 毎に必要なデータが分散配置されている。

- データベース管理プロセス
  それぞれのノードでのデータベースをローカルで管理するためのデータベース管理プログラムが存在する。

- データベース機能
  他のノードのデータベース管理プロセスと協調して、データベースの機能をユーザに提供する。

- 他のサイト、及び自サイトのデータの制御
  remote、local のデータの更新等の場合には、remote と local の制御が同調して機能する。

すなわちデータ、プロセス、協調機能、制御の 4 つが分散すべき要素であり、これらをどのように分散配置するかが分散データベースの設計の鍵となる。

6.2.3 分散データベースの利点、欠点

データが分散されていると言うことは、ユーザが作業しているローカルのサイトにデータを置くことを意味し、したがってデータはローカルで制御されることになる。分散データベースシステムにおいては、それぞれのノードにデータが置かれ、それぞれのノードでデータの制御し、一貫性の保証する責任を持つ。このように情報管理の権限と責任を分担することにより、全ての複製を複数のサイトに置く必要が無くなる。また、頻繁にアクセスするデータはユーザに近い場所にあるので、データアクセス効率も良くなる。さらに、複数のサイトに複製が置いてあるときは、複数のサイトで並列に処理を行なうことができ、障害時にはバックアップの働きを持たせることも可能である。

一方分散データベースは集中型のシステムに比べて、処理が複雑になる。信頼性と効率の向上のためにデータの複製を生成するが、

- データの更新は全ての複製に行なう必要がある
- データ更新中の障害発生時に、復旧後その更新の影響を直ちに調べる必要がある
- 他のサイトの動作と同期することが困難

といった欠点を生む。また、分散システム固有の周辺機器や、管理者の人件費などのコストがかかることに注意する必要がある。

6.3 World Wide Web におけるデータベース

近年のインターネットブームにより、多くの人々がインターネットを利用するようになった。それに伴い、World Wide Web の利用によるトラヒックの増加も問題視されてきている。そこでキャッシュやミラーといった情報資源の複製を利用することによって無駄な同一情報の再送を抑制し、トラヒックの軽減及びユーザへのレスポンスの向上を図る試みが進められている [5]。

6.3.1 Proxy Server

本来プロキシーサーバは Firewall 内から外部へのアクセスを中継するために考案された。中継時に漢字やプロトコル変換を行なうプロキシーサーバもある [22]。ここではプロキシーサーバのキャッシング機能を取り上げる。

キャッシングの研究は盛んに行なわれており、Harvest cache[6][10] が代表的である。既に実装も行なわれており、NetCache[2], Squid[4], Netscape Proxy Server[12] 等がある。また、複数のキャッシュサーバが連係するプロトコルとして ICP[1] [4] が提案されており、複数のキャッシュサーバの協調についても研究が行なわれている [13][14]。

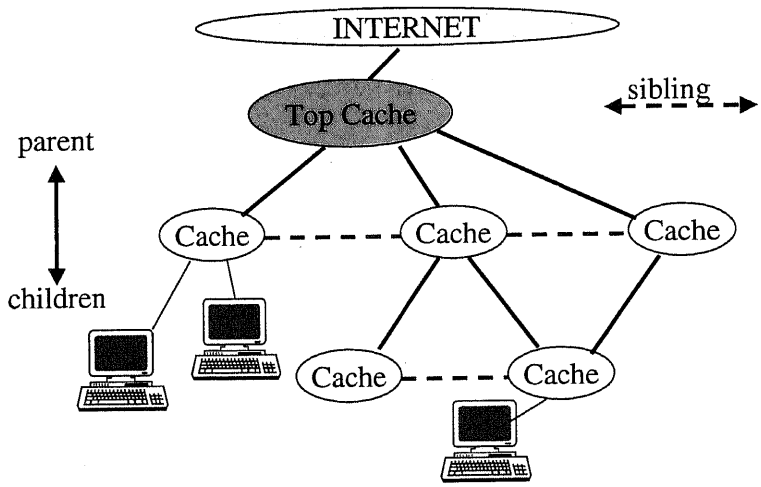


Figure 6.3: 階層化キャッシュツリー

6.3.2 キャッシュの整合性

キャッシュの整合性を表す表現として、「新鮮」「新鮮でない」という表現を用いる。次節以降ではキャッシングされたオブジェクトの新鮮さの決定法を述べる。

[4] ICP は南カリフォルニア大の Peter Danzig らによって開発され、その後 Harvest 研究プロジェクト [6] の階層キャッシュの重要な部分として改良が行なわれた。

6.3.2.1　オブジェクトの更新判定—サーバ依存の手法

キャッシュサーバでは手元にあるオブジェクトが果たして更新すべきなのかそうでないのかを判定しなくてはいけない。もし、そのオブジェクトのサーバが明示していれば判定は容易である。それを表すには 2 つの方法がある。第一の方法は Cache-Control:　max-age[5] ヘッダを用いる方法ある。このヘッダはキャッシュに残すことが許される時間を表しており、現在時間を比較して新鮮なオブジェクトかどうかを判定できる。もう一つの方法は、Expire: ヘッダを用いる方法である。Expire:ヘッダはそのオブジェクトの有効期限を示すものなので、現在時刻と比較することによって新鮮かどうかを判定することができる。

6.3.2.2　階層化キャッシュにおけるキャッシング

階層化キャッシュツリー (図 6.3) が構成されている場合について考える。最上位のキャッシュサーバを T とする。あるキャッシュサーバ C がクライアントからオブジェクト (ホームページ等)P の GET[6]要求があったとする。もし P を持っていない場合にはキャッシュサーバ C は上位のプロキシーサーバに要求を送る。そしてレスポンスを受けると、Date: ヘッダ を含むヘッダ とボディの複製を自分のディスクに格納する。ここで注意することは プロキシーサーバ C は Date:ヘッダはいじらないという点である。この Date: ヘッダ を生成するのは最上位のキャッシュサーバで、オブジェクトのサーバからそのオブジェクトをとってきた時刻が Date: ヘッダ に書き込まれる。したがって、そのキャッシュサーバより下位のサーバに存在するオブジェクトは全て同じ Date: ヘッダを持つことになる。

T では直接 P のサーバにアクセスするので、常に新鮮なオブジェクトを持っている。また、P の max-age をすべての（下位を含む）キャッシュサーバにおいて A と仮定すると T は時間 A の間は P のサーバへはアクセスしない。つまり、T はおよそ時間 A ごとに P を更新しているということになる。

次に下位のキャッシュサーバ C について考える。Date: ヘッダは最上位キャッシュサーバがそのオブジェクトのサーバから取得した時間を表すので、下位のキャッシュサーバ C においてはこの Date:ヘッダを見ることで自分のキャッシュが古いのかどうかを判定することができる。

C は常に T を経由したオブジェクトを得るので、すでにキャッシュに入っている古いオブジェクトと、それより新しいオブジェクトとの Date:ヘッダの時間差は A 以上である。したがって、T より下位のキャッシュサーバである C においても時間間隔 A より頻繁な更新処理は不要である。

6.3.2.3　下位キャッシュから上位キャッシュへの更新要求

前節においてはすべての階層化キャッシュにおいて max-age を等しく A とした。しかしながら、もし上位のキャッシュでの max-age と下位のキャッシュの max-age とが異なる値に設定されていると問題が生ずる可能性がある。下位のキャッシュより上位のキャッシュが max-age が大き

[5]HTTP/1.1[16] で導入されたヘッダである。リクエスト時には許容できるオブジェクトの古さを表し、レスポンス時にはキャッシュに残すことが許される時間を表す。

[6]Request-URI で示されたいかなる情報も、エンティティの形で取得すると言うことを意味する

くなっている場合、下位のキャッシュCがオブジェクトの更新を要求して `If-Modified-Since:`[7] を送っても、上位のキャッシュは Not Modified メッセージ[8] を返し続ける可能性があるからである。

下位のキャッシュから上位キャッシュにオブジェクトの更新を要求できる秘密は `Cache-Control: max-age=nnn` ヘッダである。クライアント（上位キャッシュから見ると下位のキャッシュはクライアントに見える）から `Cache-Control: max-age` ヘッダを受け取ると、キャッシュは現在時刻と比較した上でオブジェクトの新鮮さを検査する。すなわち、max-age=0 であれば、必ず新鮮さを調べることになる。nnn の値が小さければオブジェクトが新鮮になる可能性は高くなるが、キャッシュの効率は落ちる。一方 nnn の値が大きければ効率は上がるが、新鮮かどうかの信頼性は下がる。以下では効率の良い max-age の値の決め方を述べる。

L, D, T をそれぞれオブジェクトの `Last-Modified:`ヘッダ、`Date:`ヘッダ、現在時刻とする。また、E を消去係数とする。

このオブジェクトは時刻 $D+(D-L)E$に更新されるとすると，このオブジェクトは $T > D+(D-L)E$ が成り立てば古いということになる。よって、古さを S で表すことにすると、

$$S = T - (D + (D - L)E) \tag{6.1}$$

これを D について解くと

$$D = (T + LE - S)/(1 + E) \tag{6.2}$$

この式は次のようなことを意味している。もし、あるページを更新しようとして、`Date:` ヘッダが D_0の Not-Modified メッセージを受け取って、かつ更新したオブジェクトの古さは S_0以下になることを望む場合には

$$D_0 \geq (T + LE - S)/(1 + E) \tag{6.3}$$

となる。A_0を更新したオブジェクトの古さとして $A_0 = T - D_0$とおくと、

$$\begin{aligned} A_0 &\geq V \\ \text{但し } V &= (S_0 - E(L - T))/(1 + E) \end{aligned} \tag{6.4}$$

が得られる。

キャッシュサーバ C は `Cache-Control: max-age=V` ヘッダをつければよい。このとき、S_0はクライアントが指示してきた許容するキャッシュの古さ、あるいは求めるキャッシュの新鮮さを示す値を用いる。もしこの要求によって上位のキャッシュが Not-Modified を返した場合には、そのオブジェクトは新鮮、または許容範囲内の古さであることを意味する。

[7] このヘッダを伴った GET メソッドは条件つき GET を意味する。条件つき GET とは If-Modified-Since ヘッダで与えられる日付以降にリソースが変更された時だけ、指定されたリソースをリクエストするというものである。

[8] 条件つき GET に対して、そのリソースが If-Modified-Since フィールドで指示された日付以降に変更されていない場合にこのメッセージを返す。

6.3.3 具体例

具体的に Squid Cache[4] での設定においてオブジェクトの新鮮さの判定のアルゴリズムを説明する [3]。L,D,T をそれぞれオブジェクトの `Last-Modified:` ヘッダ、 `Date:` ヘッダ、現在時刻とし、E を消去係数とする。

そのオブジェクトの複製が生成されてからどれだけ経っているかを AGE とすると、

$$AGE = T - D$$

そのオブジェクトがどれだけ古いかを LM_AGE とすると、

$$LM\_AGE = D - L$$

LM_FACTOR は AGE と LM_AGE の比を表す。

$$LM\_FACTOR = AGE/LM\_AGE$$

CLIENT_MAX_AGE は `Cache-Control:` ヘッダの値、すなわちクライアントが許容するオブジェクトの古さを表し、EXPIRES はサーバが示すオブジェクトの有効期間である。これらの値を設定ファイル中のパラメータと比較してオブジェクトの新鮮さを判定する。

パラメータは MIN_AGE、PERCENT、 MAX_AGE の 3 つがあり、図 6.4のようなアルゴリズムによって、オブジェクトが新鮮かそうでないかを判定する。新鮮度を決定する全てのパラメー

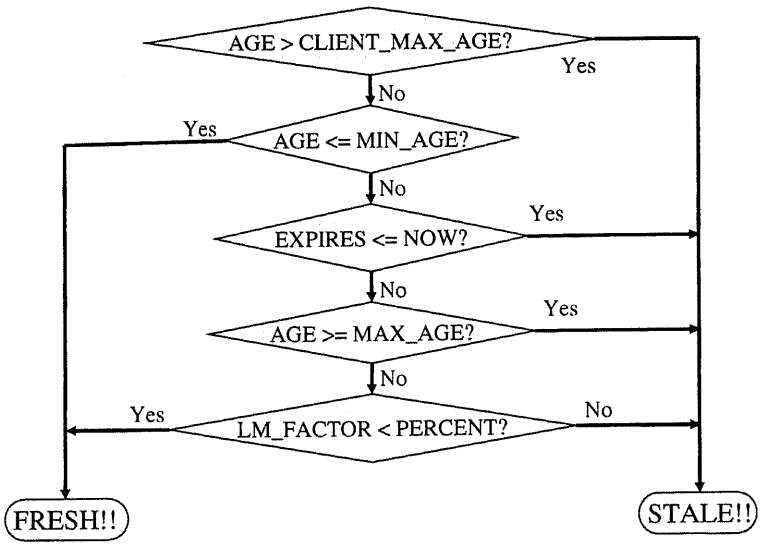


Figure 6.4: オブジェクトの新鮮度のチェック

タの優先順位は CLIENT_MAX_AGE、MIN_AGE、EXPIRES、MAX_AGE、PERCENT である。

現在求めているオブジェクトが sibling キャッシュサーバ X に存在して、その X の更新ポリシーが自分より緩やかな場合でも、オブジェクトの取得要求を X に出す時に Cache-Control: max-age を用いることで新鮮でないオブジェクトが送られてくる問題を回避できる。

6.3.4 キャッシュサーバを用いたことによるレスポンス変化

多くのキャッシュサーバの性能評価においてはヒット率が用いられている。プリフェッチを用いた Wcol[17] プロキシーサーバを用いると 3 倍程度のトラヒックによって、90%以上のホームページの先読みが可能であるとしている。当研究室では 1997 年 1 月から squid cache のログを取り続けている。今回ヒット率ではなくレスポンスに着目し、3 日間のログを集計し簡単な統計処理を行なった。

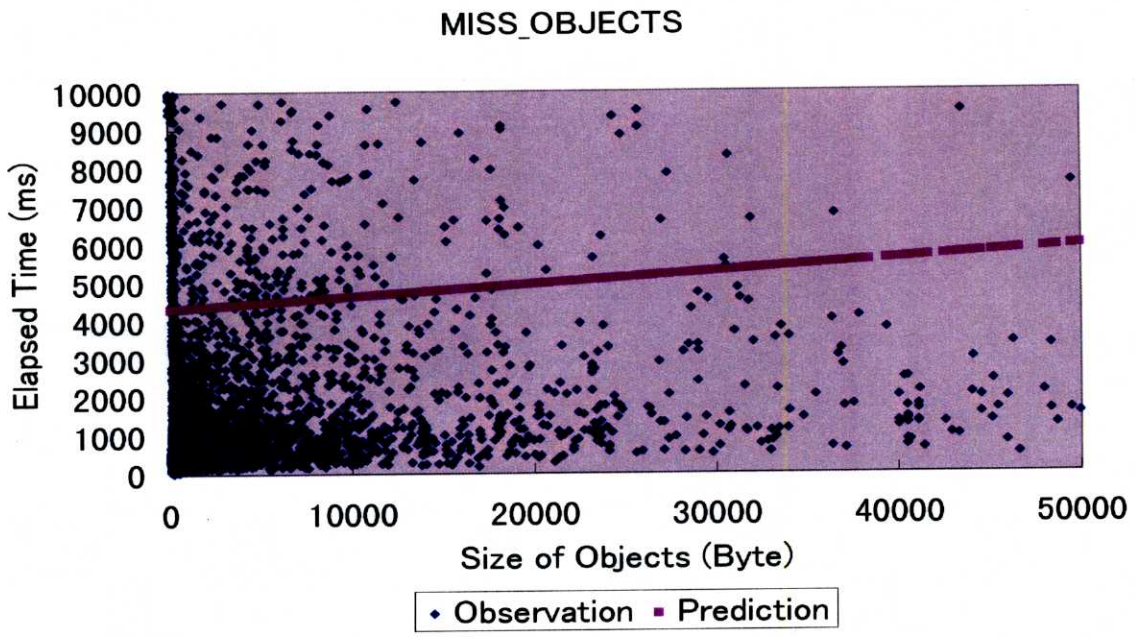


Figure 6.5: キャッシュにヒットしなかった場合

まず、キャッシュに HIT しなかった場合と、キャッシュに HIT した場合のオブジェクトサイズに対するレスポンスを表したグラフが 図 6.5,6.6 である。

グラフから明らかなように、キャッシュに HIT するとほとんどのオブジェクトが 1 秒以内に取得できていることがわかる。オブジェクトサイズ 1Byte あたりのオブジェクトの取得時間に直したもののヒストグラムが図 6.7,6.8 である。この図からもキャッシュによってレスポンスが向上したことが明らかにわかる。

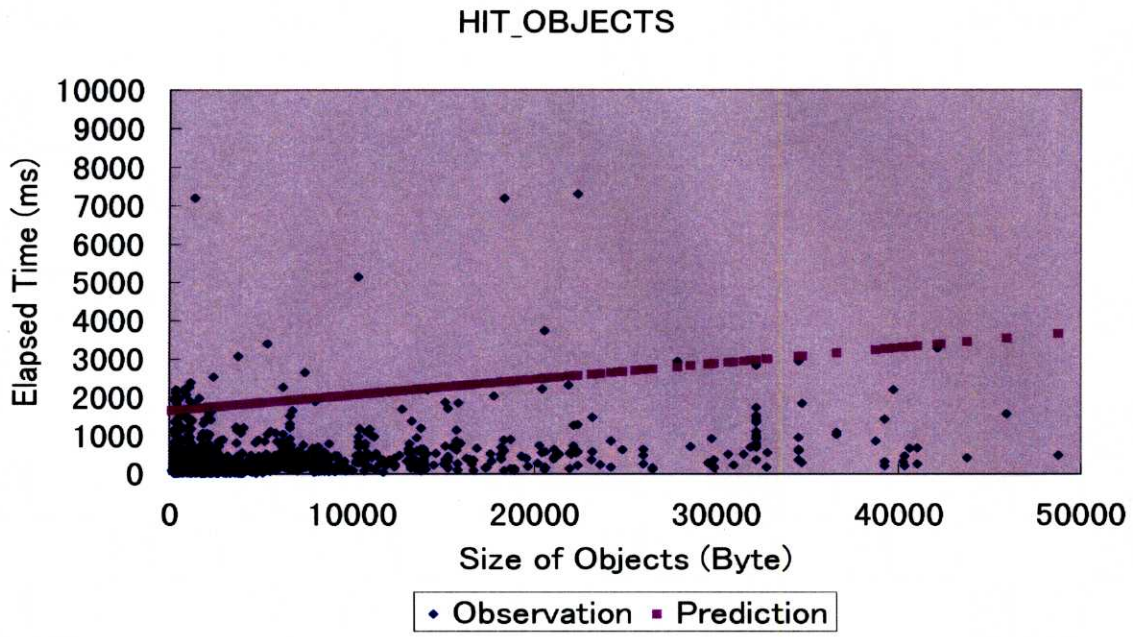


Figure 6.6: キャッシュにヒットした場合

6.3.5 親キャッシュサーバの経路による変化

東大内では 大型計算機センターでキャッシュサーバを挙げており当研究室では Parent キャッシュとして接続している。しかし、東大から海外へのアクセスは SINET を経由するため遅くなる場合が多い。そこで、省際研究情報ネットワーク (IMNET) で立ち上げられているキャッシュサーバも Parent キャッシュとして接続し、大型計算機センター経由の場合とレスポンスの比較を行なった。

まず、Parent キャッシュで HIT した場合が図 6.9 である。

HIT している場合にはあまり差が見られない。一方 Parent キャッシュHIT しなかった場合には、省際研究情報ネットワークを用いた場合の方がレスポンスが速い方に偏っていることがわかる (図 6.10)。このことから、複数の親キャッシュサーバを用いると経路の分散を図ることが可能であると言える。

6.3.6 Proxy Cache の事例

当研究室では次のようなスペック[9]のキャッシュサーバを立ち上げて利用している。

[9] このスペックは実験当時のものであり、現在は Sun Sparc 20 Workstation, Disk Swap 1G, Squid-2.2STABLE3 を使用している

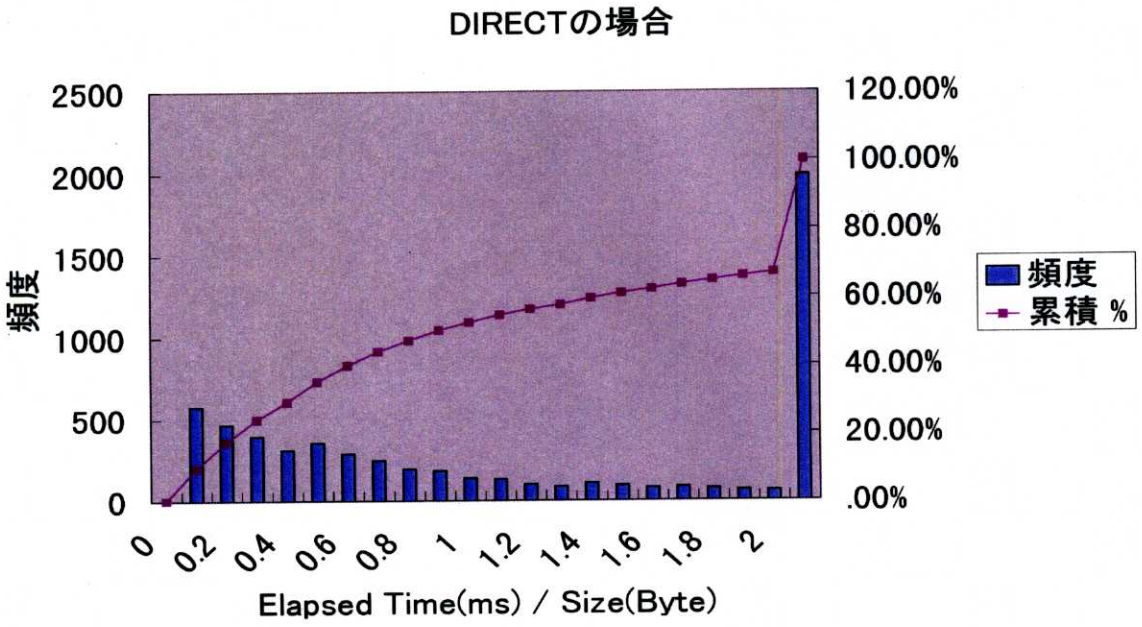


Figure 6.7: 研究室内で HIT しなかった場合のレスポンス

- Sun Sparc 5 Workstation
- Disk Swap 1G
- Squid 2.0 Patch level 2

大型計算機センター、APAN Tokyo Root Cache server[8]、電総研の Proxy Cache Server へは parent 接続、斉藤・相田研究室、坂内研究室、青山・森川研究室、教育用計算機センター、計数工学科、教養学部等とは sibling 相互接続を行っている。ここで 1998 年 4 月から 8 月までのアクセスに関してテキストデータと思われるデータ[10]でキャッシュされたユニークな URL 数を調べたものが図 6.11である。この図から羽鳥研内部からのアクセスに関しては月に約 10000URL 取得されていることがわかる。さらに他のサイトからのアクセスに関して同様のものを調べてみたものが図 6.12,6.13である。これらのサイトからのアクセスに関するヒット率は非常に低いため、互いのキャッシュにはかなり異なるものが蓄積されており、かつテキストと思われるものだけでもかなり多くのものが蓄積されていることがわかる。従って、ここにあげたサーバが連携しただけでもユーザのアクセスだけで月に約 35000URL を収集する WWW ロボットに匹敵するデータが蓄積されているのである。

[10]集計した URL は、MIME type が text/* であるか、または URL の末尾が (txt|text|htm(|l)) であり、且つ URL 中に/cgi-bin/,? を含まないものである。

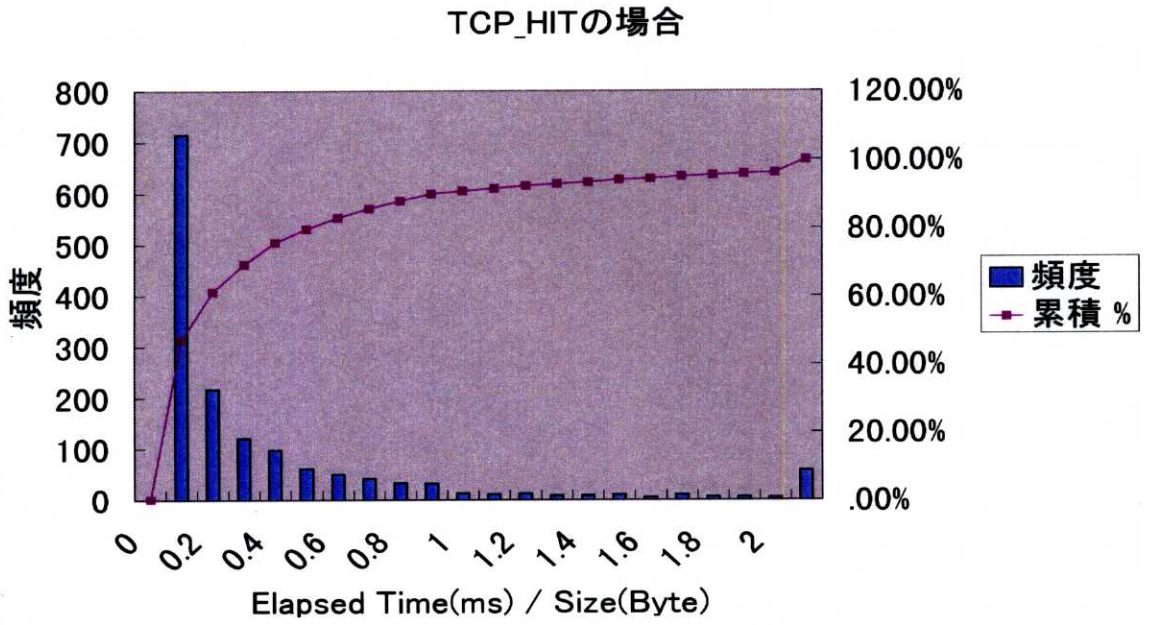


Figure 6.8: 研究室内で HIT した場合のレスポンス

6.4 モバイルエージェントと検索システム

本節ではモバイルエージェントと検索システムとの関連について検討する。モバイルエージェントを用いた複数検索システムの連携に関しての詳細は次章で述べることにする。

はじめに全文検索システムについて概観する。

6.4.1 全文検索システム namazu

「Namazu」は、高林哲氏[11]が作った全文検索システムである [27]。手軽に使えることを第一の目標とし、CGI としての動作で WWW 全文検索システムの構築が可能で、また Mail や News といった、ディスク内のファイルを対象とした個人ユースの用途にも利用可能である。

Namazu の特徴としてはいろいろあるが、以下にいくつか紹介する [29]。

マルチプラットフォーム対応 Namazu は SunOS, Solaris, Linux, FreeBSD, IRIX, HP-UX, OSF/1 などの UNIX 系のプラットフォームで動作するばかりでなく、MS-Windows95,98,NT, OS/2 等へ移植され、バイナリ配布されている[12]。

高速検索 grep 検索方式ではなくあらかじめインデクシングを行っているため検索が高速である。

[11] 現在 奈良先端科学技術大学 在学

[12] Windows 関連へは NEC エンジニアリングの広瀬健一氏、OS/2 へは住友林業の清水和佳氏によって移植されている。

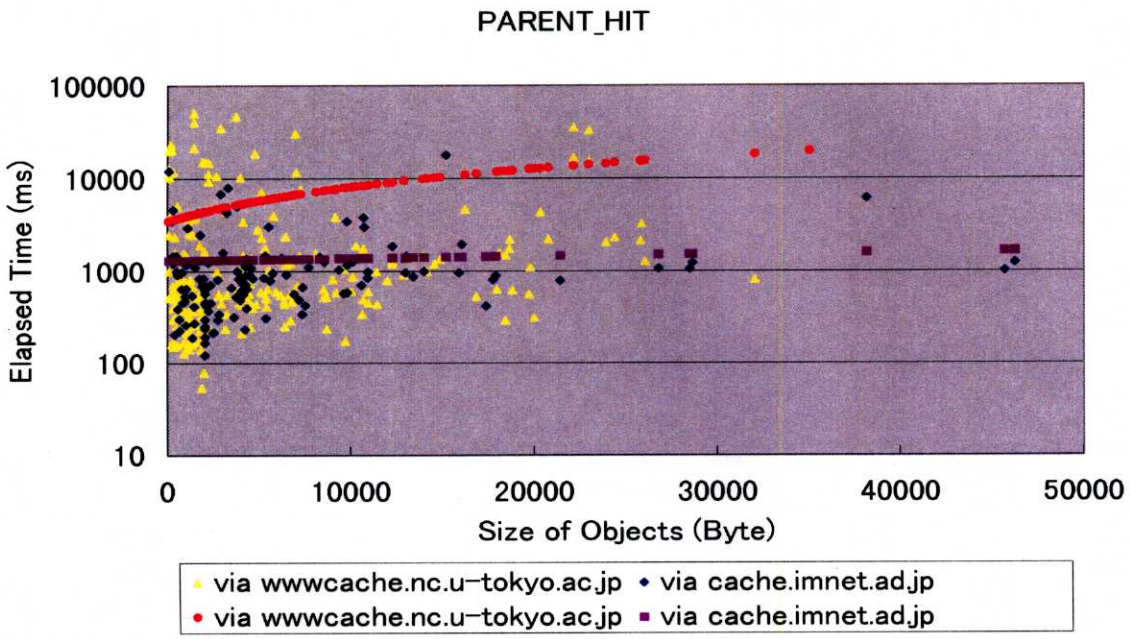


Figure 6.9: Parent Cache の比較:キャッシュHIT した場合

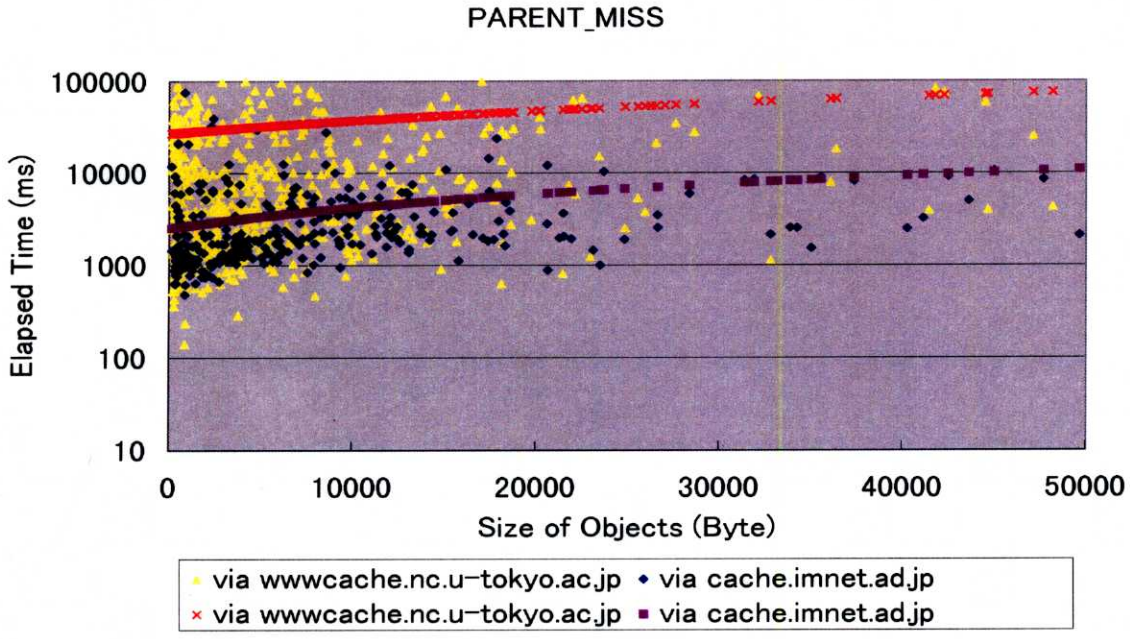


Figure 6.10: Parent Cache の比較:キャッシュHIT しない場合

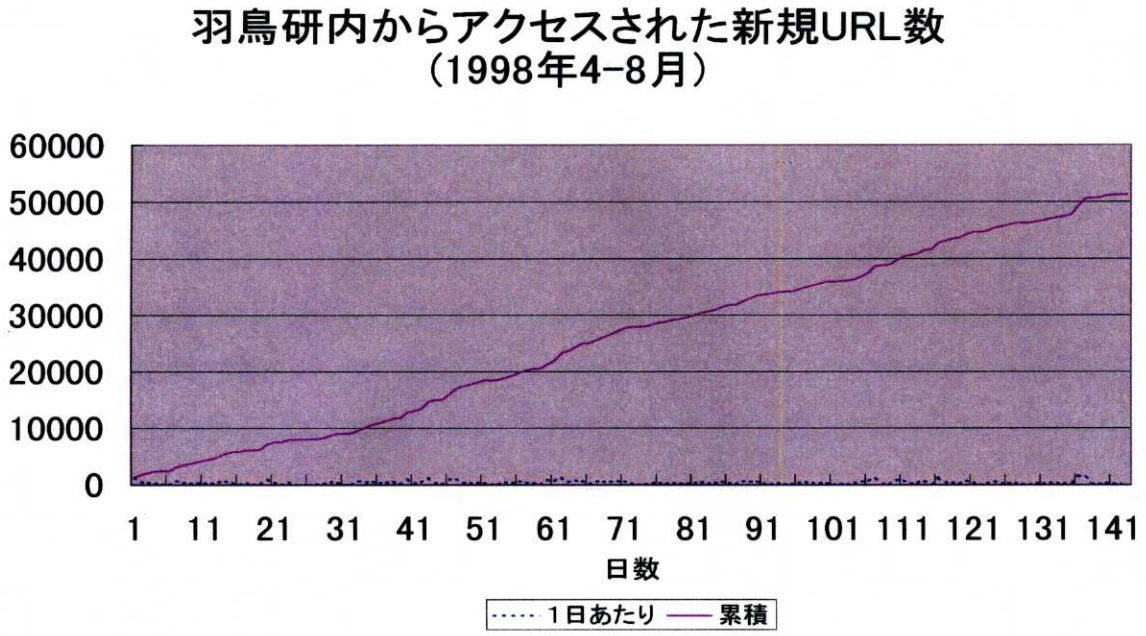


Figure 6.11: ユニークな URL 数の推移

多様な検索方法をサポート AND,OR,NOT 検索、フレーズ検索、中間・後方一致、正規表現検索をサポートしている。

複数インデックスからの検索が可能 あらかじめインデクシングを別々に行っておき、検索時にはそれぞれ別々、あるいは複数のインデックスデータに対して検索をすることが可能である。

多彩な検索クライアントの存在 コマンドライン検索、CGI、mule から呼び出して検索ができる namazu.el、Tcl/Tk 版の TkNamazu、Perl 版の pnamazu[30], Java 版の jnamazu 等がある。

これらのような特徴から、Namazu は手軽に小中規模の検索システムを構築するには非常に有効なプログラムである。したがって、Namazu で構築された小規模の検索システムが今後非常に増えてくることが期待できる。このような小規模の多くの検索システムを連携することができれば、それぞれのサイトで構築した検索システムを大規模な検索システムへの窓口として利用することが可能である。

複数の検索システムの連携については 7.3 で述べる。

6.4.2 WWW データ収集への Proxy Cache の利用

6.3で述べたように、インターネット上への WWW トラヒックの軽減のために Proxy Cache の利用が進められている。Proxy Cache のキャッシュデータはユーザがブラウズしたもののだけ

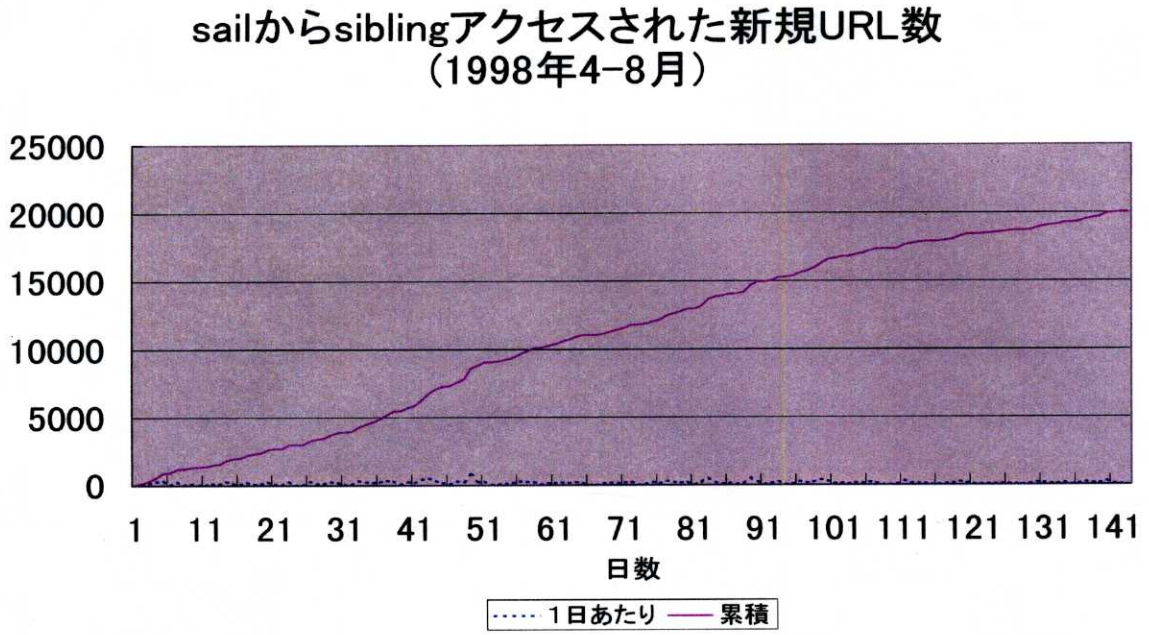


Figure 6.12: sibling hosts からのユニークな URL 数の推移 1

が蓄積したものであるが、その量は意外と多いものである。前にも述べたように、データのインデクシングのための WWW ロボットによる情報の収集は非常に多くのトラヒックを発生させる。そこで、ロボット収集において Proxy Cache を利用することが検討されている [24]。これは Proxy Cache に要求があったデータについてインデクシングを行うというものである。この方法には次のような利点がある。

- 当然ながらインデクシングに使うデータはすでに Cache の中にあるものであるから、情報収集によるトラヒックを発生させない。
- Cache の中にあるデータはユーザがある興味を持って取得したものであるから、全く無駄なものではない。また、過去にアクセスしたページをもう一度ブラウズしたくなるようなことはよくあることであり、必要なページをすぐに探し出すのはブラウザのブックマークだけでは不十分であることは多くのユーザが体験することである。
- キャッシュサーバはアクセス時にコンテンツの新鮮さをチェックするため、log を監視すれば更新したコンテンツを検出することが可能である。更新チェックのために大きなトラヒックを発生させる必要がない。

しかしながら欠点としては、過去にユーザがブラウズしたものしか取得できないという点がある。

研究室レベルであればほかの研究室のキャッシュとのヒット率があまり高くないことから、研究室ごとに蓄積されているキャッシュのデータはそれほど重複していないことが予想される。従っ

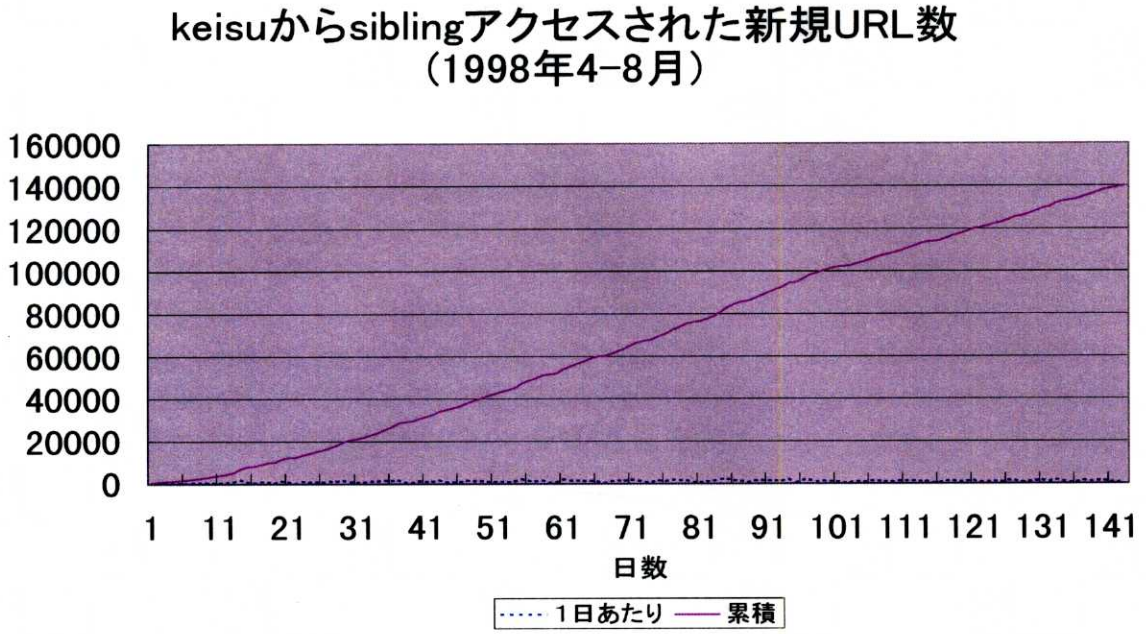


Figure 6.13: sibling hosts からのユニークな URL 数の推移 2

て、それぞれの研究室のキャッシュのデータを元にインデクシングを行い、その結果を連携して共用するようにすれば非常に効率よく比較的大きな検索システムの構築の可能性が期待できる。[24] ではキャッシュを使ったロボット同士が連携することも検討されているが、ここでは検索サーバが連携することを考えているために情報収集ロボット同士が連携する必要はない。

6.4.3 WWW データ収集へのモバイルエージェントの応用

WWW サーバ上、あるいは同じサイト内にモバイルエージェントが動作可能なプラットフォームが存在するノードがあると仮定すると、WWW データを収集する際に、自サイトでロボットを動作させる代わりにデータ収集を行うモバイルエージェント (以後収集エージェントと呼ぶことにする) をサーバ側にあるエージェント実行可能ノードに送りつけることができる。モバイルエージェントはサーバ側でデータを収集し、最終的にインデックス情報だけを自サイトに持ち帰ることによって、すべての WWW データを収集するよりもトラヒックが軽減できると考えられる。

一方、複数の検索システムが収集エージェントをサーバに送ることによって、現在以上にサーバ側に負荷をかける可能性がある。

Figure 6.14: モバイルエージェントを用いた WWW データ収集

6.4.4 WWW Search Engine へのモバイルエージェントの応用

多くの検索システムでは独立にロボットを用いてデータの収集を行っている。検索システムのデータベースを最新に状態に保つためには、頻繁にロボットでのデータ収集が必要である。しかし、各検索システムがそれぞれ頻繁にロボットを動かすことはネットワークトラヒック及びサーバ負荷の増加をもたらすことになる。そこで各サイトでサイト内のデータの検索システムを提供し外部からはメタサーチシステムを用いると、トラヒックやサーバの負荷を減少させることができると考えられる。ここで問題なのは、各検索システムの出力形式はそれぞれ異なる可能性があることで、出力を単純にマージするのは難しい。また、ユーザに提示する際には各検索システム間の重複を除去する必要がある。したがって、メタサーチエンジンと検索サーバ間の複数回のインタラクションが生ずる可能性がある。

このような場合には、モバイルエージェントを用いることによって効果的な検索が期待できる。詳細は次章で述べる。

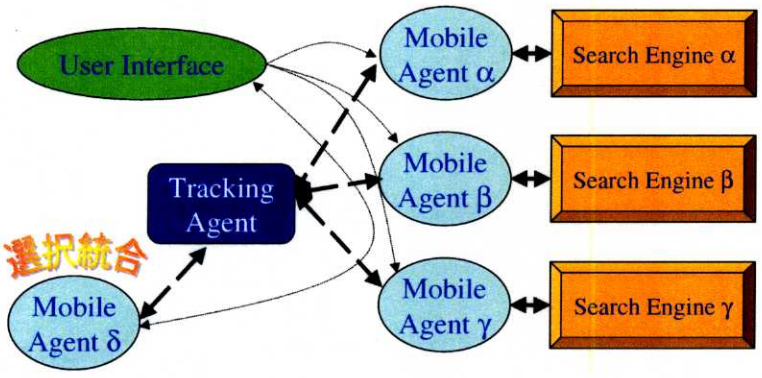


Figure 6.15: モバイルエージェントを用いた検索サーバの連携

Bibliography

[1] Internet Cache Protocol Specification. http://excalibur.usc.edu/icpdoc/icp.html.

[2] NetCache Proxy Server Software. http://www.netapp.com/products/level3/netcache/datasheet.html.

[3] Release Notes for version 1.1 of the Squid cache. http://squid.nlanr.net/Squid/1.1/Release-Notes-1.1.txt.

[4] Squid Internet Object Cache. http://squid.nlanr.net/Squid/.

[5] the Cache Now! campaign. http://vancouver-webpages.com/CacheNow/.

[6] The Harvest Information Discovery and Access System. http://harvest.transarc.com/.

[7] 日本の烏賊マップ. http://w3.lab.kdd.co.jp/proxy-jp/squid-map.html.

[8] APAN Tokyo Root Cache server. http://cache.jp.apan.net/.

[9] David Barnes and Neil Smith. An analysis of World-Wide Web Proxy Cache performance and its application to the modelling and simulation of network traffic. Technical report, University of Kent, Computing Laborator, 1996. ftp://unix.hensa.ac.uk/pub/misc/ukc.reports/comp.sci/reports/9-96.ps.Z.

[10] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, Michael F. Schwarz, and Duane P. Wessels. Harvest: A Scalable, Customizable Discovery and Access System. Technical Report CU-S-732-94, Department of Computer Science University of Colorado - Boulder, August 1994. ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Harvest.Jour.ps.Z%.

[11] E.F. Codd. A Relational Model of Data for Large Schared Data Banks. *Communications of the ACM*, Vol. 13, pp. 377–387, 1970.

[12] Netscape Communications Corporation. Netscape Proxy Server. http://www.netscape.com/comprod/proxy_server.html.

[13] John Harrison Dean Povey. A Distributed Internet Cache. In *the 20th Australasian Computer constructed which use special client software Science Conference*, Feb. 1997. http://psy.uq.edu.au/~dean/project/.

[14] Adam Dingle. Cache Consistency in the HTTP 1.1 Proposed Standard. In *ICM Workshop on Web Caching*, Sep. 1996. http://w3cache.icm.edu.pl/workshop/talk4/.

[15] Adam Dingle. Web Cache Coherence. In *Fifth International World-Wide Web Conference*, Vol. 28 of *Computer Networks and ISDN Systems*, p. 907, Paris, France, May. 1996. http://sun3.ms.mff.cuni.cz/~dingle/WebCoherence.html.

[16] Network Working Group. Hypertext Transfer Protocol - HTTP/1.1, Jan. 1997.

[17] Ken ichi Chinen. Wcol: WWW Collector Home Page —The Prefetching HTTP proxy server—. http://shika.aist-nara.ac.jp/dummy/products/wcol/wcol.old.html.

[18] Ken ichi Chinen and Suguru Yamaguchi. The Feasibility and Performance Studies on Prefetching Technique in the WWW Platform. In *Japan Internet Conference '96*, Jul. 1996. http://shika.aist-nara.ac.jp/member/k-chinen/publications/jiconf96.ps.

[19] Neil Smith. What can Archives offer the World Wide Web. Technical report, University of Kent, Computing Laboratory, 1994. ftp://unix.hensa.ac.uk/pub/misc/ukc.reports/comp.sci/reports/11-94.ps.Z%.

[20] Hal Stern. *NIS & NIS*. 株式会社アスキー, 1992.

[21] Duane Wessels. SQUID Frequently Asked Questions. http://squid.nlanr.net/Squid/FAQ/FAQ.html.

[22] ysato@etl.go.jp. What is DeleGate? http://ringer.etl.go.jp/ysato/DeleGate/.

[23] 内藤広志. squid を用いた学内 proxy 網の構築構築. 第 8 回メディア統合技術研究会オンライン予稿集. 画像電子学会, 1997. http://www.oit.ac.jp/ij/naitoh/mitc/mitc8/mt8s31/index.htm.

[24] 吉岡恒夫. 代理サーバを利用した検索システム. http://infonet.aist-nara.ac.jp/~tsuneo-y/proxysearch/.

[25] 西尾章治郎. 実践 SQL 教科書. アスキー出版局, 1996.

[26] 滝沢誠. モバイルデータベースシステム. 電子情報通信学会誌, Vol. 80, No. 4, pp. 331–337, Apr. 1997.

[27] 高林哲. 全文検索システム Namazu. http://www.ring.gr.jp/openlab/namazu/index-j.html.

[28] 奥山徹. 分散データベース概説その 3, Nov 1994.

[29] 馬場肇. 日本語全文検索システムの構築と活用. ソフトバンク, Sep. 1998.

[30] 古川令. namazu 検索クライント perl 版 (pnamazu). http://saturn.aichi-u.ac.jp/~ccsatoru/Namazu/pnamazu.html.

Chapter 7

モバイルエージェントによるWWW連携検索

最近WWWの検索技術を利用した小規模な検索システムが普及し始めている。本章ではそのような小規模な検索システムをモバイルエージェントを用いて連携し、グローバルな検索システムへの窓口を実現することを目指す。

7.1 WWWにおける検索システム

WWW空間は情報の宝庫であるけれども、あまりに広大で求める情報にいつもすぐにたどり着けるとは限らない。そのような場合に、我々にとってWWW空間における地図ともコンパスともいえるものがWWW検索サービスである。しかしながら、このWWW検索サービスの構築のためには多くの計算機パワー、ディスクを消費し、そして大きなトラヒックを発生させる場合がある。

さて、最近手軽に小中規模の全文検索システムが構築できるNamazu[8]と呼ばれるシステムが注目され、現在ユーザが増えつつある。このような小規模の検索サービスが連携することによって、比較的大きな検索サービスが実現できるのではないだろうか。本章ではこのような小規模の検索サーバの動的な連携について検討を行っている。検索サーバ間の連携は静的でも良いだろうが、本章ではモバイルエージェントを用いた連携を検討している。

筆者はこれまで、協調する複数のモバイルエージェント間の通信についての新しい枠組みを提案してきた[3,6]。現在モバイルエージェントは、単独で仕事を行い、その通信相手は固定であるとして扱われることが多い。しかしながら、今後は多数のモバイルエージェント間の通信が必要となる場面が増加すると考えられる。モバイルエージェントを用いた検索サーバ間の連携は、そのアプリケーションの一つになるものとして着目している。

7.1.1 WWW 検索システム

現在、世界には 200 万を越える WWW サーバが存在するといわれている [4]。WWW はインターネット上での情報提供、収集の手段として世界中で利用されている。しかしながら、情報収集においては必要な情報を自力で探し出すことは非常に困難である。そこで AltaVista[1] や Yahoo[5] といった検索システムを利用することになる。

検索システムはジャンルやテーマに沿って分類されたリンク集といえるカテゴリ型と、ユーザがキーワードを用いて検索を行うロボット型の二つに大きくわけることができる。本論文では検索システムはロボット型を指すものとする。

7.2 モバイルエージェントの検索システムへの応用

WWW 空間の増加の速度は非常に大きく、現在単独の WWW ロボットによる収集では追い付かなくなっているといわれる。そこで収集能力を補うために他の検索システムの検索結果を利用するメタサーチサービスを構築する方法がある。

ここで問題なのは、各検索システムの出力形式はそれぞれ異なる可能性があることで、出力を単純にマージするのは難しい。また、ユーザに提示する際には各検索サービス間の重複を除去する必要がある。したがって、メタサーチエンジンと検索サーバ間の複数回のインタラクションが生ずる可能性がある。

このような場合には、検索サーバにモバイルエージェントが動作することのできるプラットフォームを置き、複数のモバイルエージェントを同時に各検索サーバに送り、モバイルエージェント間で連携して重複を除去した上で、各検索サーバからの結果を返すようにすることができよう。

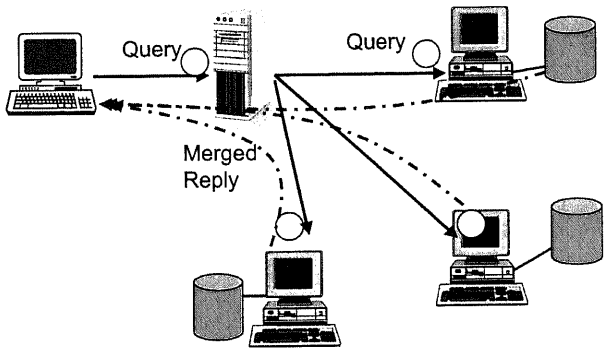


Figure 7.1: モバイルエージェントを用いた検索サーバの連携

7.3 複数の検索サーバの連携

一つ一つの検索サーバの規模は小さくても、互いに連携することでより大きな規模の検索サーバとなることが期待できる。

7.3.1 小規模検索システム

近年 CPU や disk の性能は向上し、学生のホームディレクトリが GB 単位で割り当てられるようにもなった。また我々の身近にもノートパソコンを持ち歩く人を多く見掛けるようになり、持ち歩く情報の量が増大してきている。

このようなディスクの容量の増加にともない、メール、ニュースといったプライベートな情報の量も増大している。自分の過去に受信したメールの中からあるメールを探し出したいようなときなど、一つ一つ目でみながら探すことはとても不可能になりつつある。

このような場合に unix の grep コマンドで探すことも可能であるが、全文検索システムがあると非常に便利である [9]。これまでの全文検索システムは規模が大きくインストールに手間もかかり、とても小規模なプライベートの情報を扱うには割が合わなかった。ところが最近 Namazu といった プライベートユースにも手軽に使える全文検索システムが登場し、注目されつつある。

プライベートな情報は今後もますます増えていくと考えられるので、このような手軽な全文検索システムは非常に有効であろう。

7.3.2 イントラネットでの活用

イントラネット環境において、各部署毎に全文検索システムを構築し全社内で互いに連携するとする。すると、検索をする際には全社のドキュメントから検索を行うことができるようになるのである。また、全社のデータベースの更新は規模が大きいために時間もかかるが、部署単位で検索システムが構築されていれば、自分の部署の中の文書が更新された場合だけ部署内の検索システムのデータベースを更新するだけで良いというメリットもある。

7.3.3 キャッシュサーバとの連携

6章でも述べたように、インターネット上の WWW トラヒックの軽減のために Proxy Cache の利用が進められている。Proxy Cache のキャッシュデータはユーザがブラウズしたもののだけが蓄積したものであるが、当研究室では Proxy に蓄えられる新規の URL のうちで、インデクシングが可能と思われる URL 数は約 10000/月である (6.3.6)。データのインデクシングのための WWW ロボットによる情報の収集は非常に多くのトラヒックを発生させる。そこで、ロボット収集において Proxy Cache を利用することが検討されている [7]。これは Proxy Cache に要求があったデータについてインデクシングを行うというものである。この方法には次のような利点がある。

- インデクシングに使うデータはすでに cache の中にあるものであるから、情報収集によるトラヒックを発生させない。
- cache の中にあるデータはユーザがある興味を持って取得したものである可能性が高い
- キャッシュサーバはアクセス時にコンテンツの新鮮さをチェックするため、log を監視すれば更新したコンテンツを検出することが可能である。更新チェックのために大きなトラヒックを発生させる必要がない。

しかしながら欠点としては、過去にユーザがブラウズしたものしか取得できないという点がある。

研究室レベルであればほかの研究室のキャッシュとのヒット率があまり高くないことから、研究室ごとに蓄積されているキャッシュのデータはそれほど重複していないことが予想される。従って、それぞれの研究室のキャッシュのデータを元にインデクシングを行い、その結果を連携して共用するようにすれば非常に効率よく比較的大きな検索サービスの構築の可能性が期待できる。[7] ではキャッシュを使ったロボット同士が連携することも検討されているが、検索サーバが連携すれば情報収集ロボット同士が連携する必要はない。

7.3.4 検索サーバ間連携プロトコル

検索結果をユーザに提示する場合には、優先度を決定するスコアリングという処理が必要である。namazu を含めた多くの検索サーバでは、検索結果のスコア付けに tf·idf 法またはその変種を用いている。tf·idf 法は多くの文書に含まれるキーワードよりも数少ない文書に含まれるキーワードの方のスコアを高くする方法で、複数にキーワードを用いた検索に有効である。もし、tf·idf 法を用いるサーバ同士が連携した場合、複数の検索サーバの検索結果を単にスコアの高い順にマージソートしたのでは正しいマージ結果を得ることはできない。

このようなことを避けるためには、サーバ連携時には 2 パスのスコア付けが必要である。そこで、次のようなサーバ間連携プロトコルを提案する。

Query の要求を出すものを C(Search Client) 、検索結果を返すもの (複数) を S(Search Server) とする。また、ユーザが要求する URL 数を N とする。

1. C は S に query 要求をする。その際、検索結果としては URL のみを要求し、さらに各キーワードの出現頻度も要求する。S の数を S_n とすると、各サーバに対して最高 N/S_n の検索結果を要求する。
2. C は S に query 要求のキーワード、及びすべての S からのキーワードの出現頻度の総和を渡して、再スコアリングを要求する。また、得られた検索結果のうち、ある一定のスコア以下のものが欲しい場合にはそのスコア MAX も送る。
3. C は 2 で得られた S からの結果をマージする。もし、検索結果数が N 未満の場合には 2 を繰り返す。
4. もしパス 2 のレスポンスが遅い場合には、暫定的なスコアづけでソートを行い結果を提示する。

このようにすることで、tf-idf 法を用いて複数サーバからの検索結果を利用しても正しいスコ

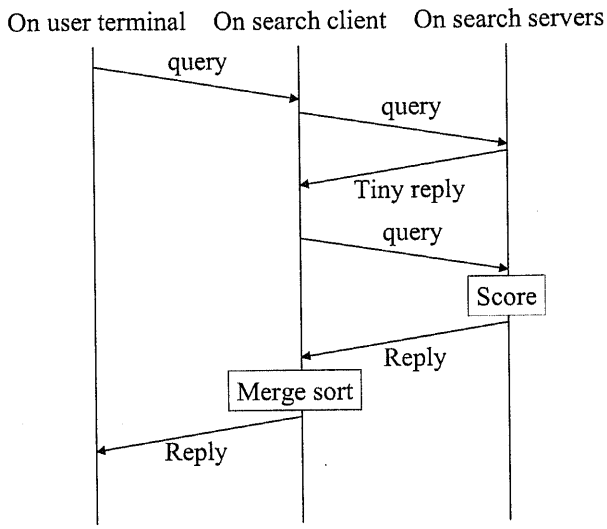


Figure 7.2: サーバ間連携プロトコルの流れ

アを得ることができる。

7.4 再帰的検索

複数の検索サーバが連携して検索を行うと、再帰的な検索が発生する。この再帰的検索においては次のような問題点が考えられる。

- 再帰的検索の深さ
- 検索遅延
- 再帰的検索ループ

以下にこれらの問題点の詳細と解決法を述べる。

7.4.1 再帰的検索の深さ、遅延

検索サーバ間が連携する場合には、一つの検索サーバに検索をかけた場合に他のサーバへの検索が実行される可能性があり、さらにそのほかのサーバへの検索が実行される可能性がある。このような再帰的検索の場合にどの深さまで実行するか、どの深さでうち切るかが問題である。さらに、検索の深さにも関連するが、再帰的に検索を行うことによって遅延が大きくなってしまう可能性がある。ユーザによっては遅くてもいいからなるべく多くの検索結果がほしい場合や、多少少なくてもいいから素早い検索結果がほしい場合等が考えられる。これらの問題の対策の一つとして、ユーザから階層またはタイムアウトを指示するという方法が考えられる。それぞれの利

点・欠点は以下の通りである。

1. タイムアウト指定
   指定されたタイムアウトにしたがって検索を打ち切るとすると、ユーザへのレスポンスの向上を期待できる。しかしながら、打ち切るホストの数が多いと、それぞれのホストへの負荷、及びネットワークトラヒックの無駄が生ずることになる。
2. 階層の深さの指定
   ユーザによって階層の深さを指定することで、検索による負荷のかかるホストを限定することができ、また再帰的検索のトラヒックの発生する範囲を限定することができる。しかしながら階層の深さを指定しても必ずしもユーザへのレスポンスに反映されるとは限らない。

以上の理由から、再帰的検索においてユーザまたはユーザエージェントが検索の条件を指定する場合には、階層の深さ、及びタイムアウトの両方に関する指定をする必要があるといえる。

7.4.2 ループ検出

2 階層以上の再帰検索を行う場合、検索がループしてしまう可能性がある。ループを防ぐためには次のような方法が考えられる。

1. 検索リクエストを行う際に、検索リクエストを発行したクライアントの情報、セッション ID を付加する
2. 連携する際に組織化を行う

1は検索サーバ上で履歴を残すことによってループを検出することができる。一方 2は、あらかじめ連携している検索サーバ同士がループを生じないように組織化を行うという方法である。

7.4.3 連携の形態

連携した複数サーバには、二つ利用形態が考えられる。一つは常に全連携検索サーバから検索を行いたい場合、もう一つはユーザ側の示す条件下でなるべく多くの結果を取得したい場合である。前者はたとえばイントラネット内の検索のような場合が考えられる。このような場合は、あらかじめ検索サーバ同士をツリー構造のように階層化して組織化しておいた方が効率が良いと考えられる。一方後者は、例えばミーティングに集まった個人のノートパソコンを連携するような場合が考えられる。このような場合には、負荷に応じて動的に連携の構造を変化させていくと良いと考えられる。

7.4.4 再帰的検索に要する時間

図 7.3のように、ユーザはユーザエージェントを通して検索サーバ A に検索要求を出し、検索サーバ A は検索サーバ B,C に再帰的に検索を行うとする。このとき、7.3.4章で述べた 2 パスのサーバ間連携プロトコルを用いるものとする。

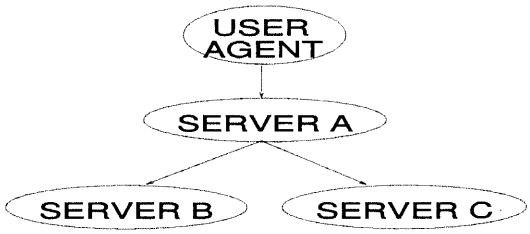


Figure 7.3: A Hierarchy Search System

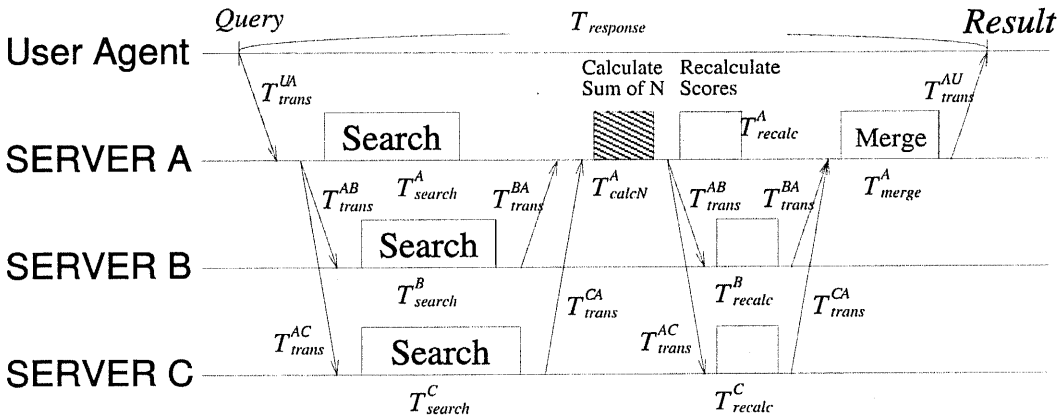


Figure 7.4: 再帰的検索の流れ

ここで、User Agent、検索サーバA,B,CをそれぞれU, A, B, Cと表わす。図7.4のようにT_{trans}はデータを転送にかかる時間で、T^{AC}_{trans}はサーバAからCへの転送時間である。検索、スコアの再計算、結果のマージにかかる時間をそれぞれT_{search}, T_{recalc}, T_{merge}と表わす。また、サーバAが連携プロトコルパス2のためのキーワード出現頻度の総和、全文書数の総和を計算する時間をT^{A}_{calcN}, ユーザエージェントに結果を返すまでの時間を$T^{D}_{response}$（再帰的検索をしない場合）, $T^{II}_{response}$（再帰的検索する場合）とする。

もしサーバAがサーバB,Cに再帰的検索を行わない場合は

$$T^{D}_{response} = T^{UA}_{trans} + T^{A}_{search} + T^{AU}_{trans} \tag{7.1}$$

である。サーバB,Cに再帰的検索を行うとすると

$$T^{II}_{response} = T^{UA}_{trans} + \max\left[T^{A}_{search}, (T^{AB}_{trans} + T^{B}_{search} + T^{BA}_{trans}),\right.$$

$$
\begin{aligned}
&(T^{AB}_{trans} + T^{B}_{search} + T^{BA}_{trans})\Big] \\
&+ T^{A}_{calcN} \\
&+ \max\Big[T^{A}_{recalc}, (T^{AB}_{trans} + T^{B}_{recalc} + T^{BA}_{trans}), \\
&(T^{AB}_{trans} + T^{B}_{recalc} + T^{BA}_{trans})\Big] \\
&+ T^{AU}_{trans}
\end{aligned}
\tag{7.2}
$$

となる。$T^{A}_{search} = T^{B}_{search} = T^{C}_{search} = T_{search}$, $T^{A}_{recalc} = T^{B}_{recalc} = T^{C}_{recalc} = T_{recalc}$ と仮定すると 7.2式は次のようになる。

$$
\begin{aligned}
T^{H}_{response} = \\
&T^{UA}_{trans} + T_{search} \\
&+ \max\Big[(T^{AB}_{trans} + T^{BA}_{trans}), (T^{AC}_{trans} + T^{CA}_{trans})\Big] \\
&+ T^{A}_{calcN} + T_{recalc} \\
&+ \max\Big[(T^{AB}_{trans} + T^{BA}_{trans}), (T^{AC}_{trans} + T^{CA}_{trans})\Big] \\
&+ T^{AU}_{trans}
\end{aligned}
\tag{7.3}
$$

さらに、$T^{AB}_{trans} = T^{BA}_{trans}$, $T^{AC}_{trans} = T^{CA}_{trans}$, $T^{UA}_{trans} = T^{AU}_{trans}$ とすると、

$$
\begin{aligned}
T^{H}_{response} &= 2T^{UA}_{trans} + T_{search} \\
&\quad + 4\max\Big[T^{AB}_{trans}, T^{AC}_{trans}\Big] + T^{A}_{calcN} + T_{recalc}
\end{aligned}
\tag{7.4}
$$

となる。したがって、式 7.1 式 7.4より、1 階層再帰的検索を行うごとに

$$
\begin{aligned}
\Delta_{response} &= T^{H}_{response} - T^{D}_{response} \\
&= 4\max\Big[T^{AB}_{trans}, T^{AC}_{trans}\Big] + T^{A}_{calcN} + T_{recalc}
\end{aligned}
\tag{7.5}
$$

だけ時間がかかることになる。

7.4.5 再帰的検索打ち切り決定法

本節ではユーザ、またはユーザエージェントから指定された階層、またはタイムアウト値によって再帰的検索を打ち切るためための手順を述べる。

まず、ユーザまたはユーザエージェントが検索要求をする際には、検索キーワード、希望出力件数の他に次のような情報を発信する。

- ユーザエージェントの ID
- セッション ID
- 検索最大階層

- 最大待ち時間

検索最大階層、最大待ち時間のいずれかが与えられなかったときのために、デフォルトの値を設定しておく。また、セッション ID は同一エージェントが複数の検索要求を発行した際にそれぞれの検索要求を区別するためのものである。

さて、図 7.3において、検索サーバ A が検索要求を受けたとき、再帰的検索を行わない場合には結果を返すまでにかかる時間は 7.1式で与えられる。また、1 階層再帰検索によって増加する処理時間は 7.5式である。ここで Aglets を用いて実装を行うと、メッセージのやり取りのイベントや外部プロセスの起動に時間がかかるので、$T^{A}_{calcN}, T_{recalc}$は内部計算処理だけであることから

$$T^{AB}_{trans}, T^{AC}_{trans} \gg T^{A}_{calcN}, T_{recalc} \tag{7.6}$$

であり、7.5式は

$$\Delta_{response} \simeq 4 \max \left[T^{AB}_{trans}, T^{AC}_{trans} \right] \tag{7.7}$$

となる。

予備実験において Aglets でのデータの転送にかかる時間は、数 kB 程度までならばデータのサイズに依存しないことが得られているため、7.7式の$\Delta_{response}$は検索データにほぼ依存せず、あらかじめ測定しておくか、過去の値を利用することが可能である。

検索サーバ A が検索サーバ B,C に再帰的検索を実行するのは次の条件がそろった場合である。

- ユーザから指定された再帰階層の深さを超えないこと
- $T^{H}_{response}$ が最大待ち時間以内で収まりそうであること

過去の検索での $T^{UA}_{trans}, T^{A}_{search}, T^{AU}_{trans}$ を元に $T^{H}_{response}$ を計算し、この値が最大待ち時間以内であればサーバ B,C へ検索要求をする。

一方サーバ B,C ではサーバ A からの検索要求を受けるとサーバ A からの検索セッション ID が検索履歴の中に存在しないことを確認する。もし検索履歴の中にセッション ID が存在する場合には、再帰検索のループが発生している可能性があるため再帰検索の要求を拒否する。

以上のようにして再帰検索を行う。

7.5 継続検索

ユーザが検索を行なう多くの場合、提示する検索結果の数の上限 (たとえば 10 件) を指定する。もし提示された結果の中にユーザの求めるものがなく、且つサーバには更に多くの検索結果がある場合には、ユーザはサーバに次に高いスコアを持つ結果 (次の 10 件) の提示を求めることができる。このような「次の 10 件」というような検索を本論文では継続検索と呼ぶことにする。

本節では協調検索システムの実装に当たって、この継続検索でのスコア計算において 2 パスの方式と 1 パスの方式のそれぞれの動作について比較検討を行なう。

7.5.1 継続検索におけるスコアリング

図 7.3のようなモデルを考える。検索サーバ A はユーザからの検索要求に対して自分自身で検索すると同時に検索サーバ B,C にも検索要求をし、A,B,C3 サーバの結果を統合してユーザに提示するとする。このようなモデルの場合において再帰的に検索を行なう場合の受け渡すデータについて検討する。

2 パスの場合には 7.3.4 で述べた連携検索プロトコルを用いるものとする。したがってサーバ間で受け渡すデータは以下のようになる。

一方 1 パスの場合には ServerB, C の結果、および ServerA での結果を全て ServerA 上でスコア計算を行なう必要がある。したがって、再帰検索を行なう際には次のようなデータの受渡しが必要である。

- **A→B,C:** ユーザの入力キーワード
- **B,C→A:** それぞれのサーバ上の全文書数、それぞれのキーワードを持つ文書数、マッチした結果、結果それぞれの Tf 値

さて、多くの場合ユーザがキーワード入力し検索要求をする場合、結果の数を指定する。したがって検索結果がその指定数よりも多い場合には「次の 10 件」といった継続検索を行なうことができる。このような継続検索の場合において 1 パス方式、2 パス方式それぞれの動作を検討する。

1 パス方式の場合には ServerA 上でスコア計算を行なって初めて正しいスコアが得られる。したがって、ServerB,C 上で暫定的にソートした結果の順序が必ずしも ServerA 上でスコアを再計算した場合と同じであるとは限らない。したがって、継続検索を可能にするためには、ServerA,B,C から得られた全ての結果を保持しておく、もしくは継続検索の場合にも最初の検索と同様の処理を行ない (すなわち、ServerB,C から全ての検索結果の転送をし)、全ての検索結果の中から該当する部分を取り出す必要がある。

一方 2 パスの場合には全サーバで合計した全文書数とそれぞれのキーワードを持つ文書数を保持して、パス 2 にあたる処理のみを行なえば良い。その際に、すでに提示した結果の最低スコアを ServerB,C に送ることによって、ServerB,C からは全ての検索結果を ServerA に送る必要がない。

以上を鑑みるに、1 パスの方式では継続検索を行う場合に必ず全てのサーバから全ての結果を転送する必要があるので、サーバからの結果数が多い場合には効率が良くない。

7.5.2 実装に当たって

2 パスの方式は、暫定検索、スコア再計算いずれの場合においても協調しているサーバへの再帰的な通信が生じる。しかしながら、継続検索の際には前回ユーザに提示したスコア以下のものだけを各サーバ上で容易に選別することができる。一方 1 パス方式では初回の検索では 2 パス方式に比べて速い検索が期待できるが、継続検索の際には無駄が生ずると考えられる。そこで、二つの方式を融合し、2 パスの方式において初回の検索の際に転送するデータを増やすことによって 1 パスで結果を出し、継続検索の際には 2 パス方式の 2 パス目の検索を行うことが望ましいと

考えられる。

7.6　モバイルエージェントを用いて連携した検索システムの実装

本節ではモバイルエージェントを用いた連携検索システムの実装について述べる。

7.6.1　実装形態

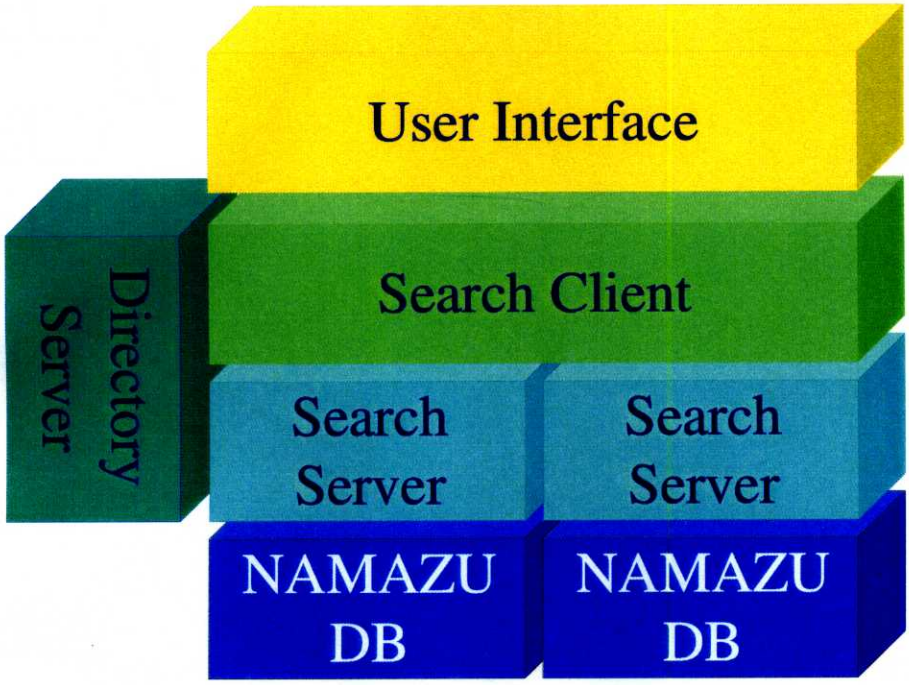


Figure 7.5: 各コンポーネントの関係

本実装においては次のように 4 つのエージェントと Namazu のデータベース、計 5 つの主要なコンポーネントがある。それぞれの関係は図 7.5のようになっている。

1. User Interface Agent (UI)
   ユーザからの検索要求を受け付け、検索結果を表示するエージェント

2. Serach Client Agent (SC)
   User Interface Agent からの検索要求を受け付け、各検索サーバに検索要求を出す。また、検索結果の統合も行う。

3. Search Server Agent (SS)
   Namazu DB にアクセスして検索を行う部分。

4. Namazu DB (DB)
   Namazu のデータベースファイルを用いる。あらかじめ Namazu の mknmz によって必要なコンテンツのデータベースを構築しておく必要がある。本実装においては Namazu の検索プログラムは使用していない。

5. Directory Server (DS)
   Search Server の位置を登録するサーバ。

実装環境は表 7.1の通りである。

Table 7.1: 実装環境

使用言語	JDK 1.1.8
モバイルエージェント プラットフォーム	ASDK 1.0.3[2]
検索システム、データベース	Namazu
検索エンジン	Java による実装

7.6.2　検索の流れ

図 7.6, 7.7, 7.8, 7.9, 7.10が実装した検索システムのウィンドウである。検索は以下のような手順で行われる。

1. Derectory Server (DS) を起動する (図 7.6)。

2. 各 Namazu DB のあるホスト上で Search Server (SS) を起動する (図 7.7)。この実装例では 4 つの SS を起動した。それぞれネットニュースグループの comp.mail.*, comp.lang.*, comp.infosystems.*, comp.database.* 以下の記事のデータベースを検索するサーバとなっている。SS は自動的に起動時に DS に自分のいるエージェントプラットホームの情報 (AgletContext) 及び 自分のエージェント ID (AgletID) を DS に登録する。DS は指定することもできるが、指定しない場合にはハードコーディングされた場所の DS への接続を試みる。図 7.6 の DS は既に 4 つの SS が DS に登録されたことをしてしている画面である。

3. SS でサービスするデータベースのディレクトリを選択する (図 7.8)。図 7.7の SS は comp.database.* の News group の記事のデータベースを指定した。

4. ユーザが User Interface Agent (UI) を起動する (図 7.10)。UI は自動的に Search Client (SC) を起動する (図 7.9)。

5. 起動した SC は DS から SS の情報を取得する。これによって SC は SS へのコネクションを張る準備ができる。

6. ユーザが UI に対して検索キーワードを入力する。図 7.10はキーワードとして “mime”, “perl” の二つを入力した。
7. UI がキーワードを SC に送る
8. SC は SS から取得した一つ又は複数の SS にキーワードを送り、検索要求を行う。マルチスレッドで行うため複数の SS に対して同時に検索要求を行うことができる。
9. 各 SS から得られた検索結果をマージし、スコアの再計算を行いソートをしなおす。図 7.9 は各サーバから得られたヒット数が示されている。
10. CS は検索結果を UI に渡す
11. UI が検索結果をユーザに提示する (図 7.11)。

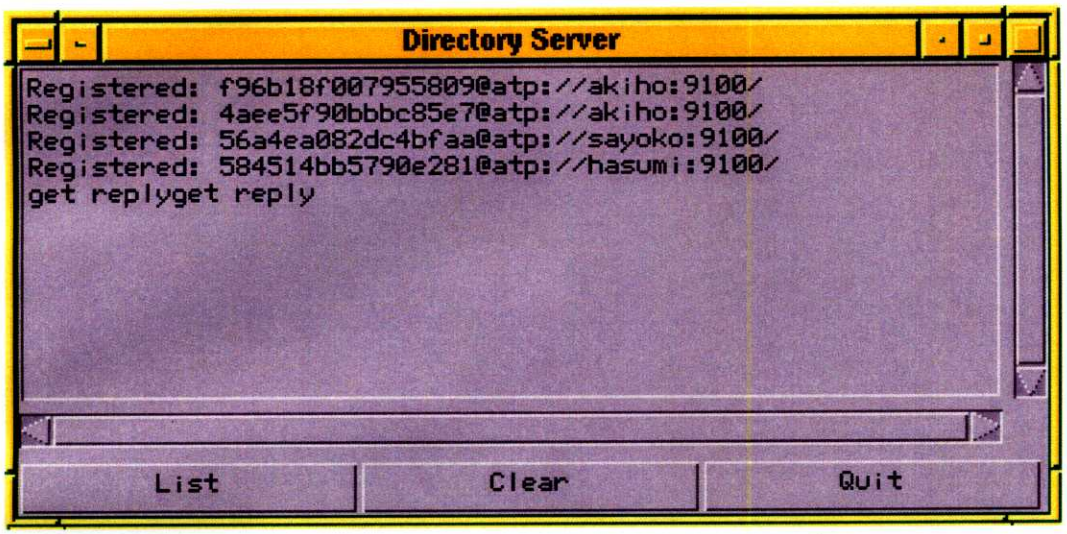


Figure 7.6: Directory Server

7.6.3　エージェントの動作の詳細

本実装においてはモバイルエージェントプラットフォームを用いているにも関わらず、Agent migration を用いている部分は少なく、主にエージェント間のメッセージ通信によって仕事を行っている。各エージェントのメッセージインターフェースについてより詳しく述べる。

- Directory Server
  メッセージの種類として “get” “put” “delete” “clear” を受け取る。“get” に対しては、

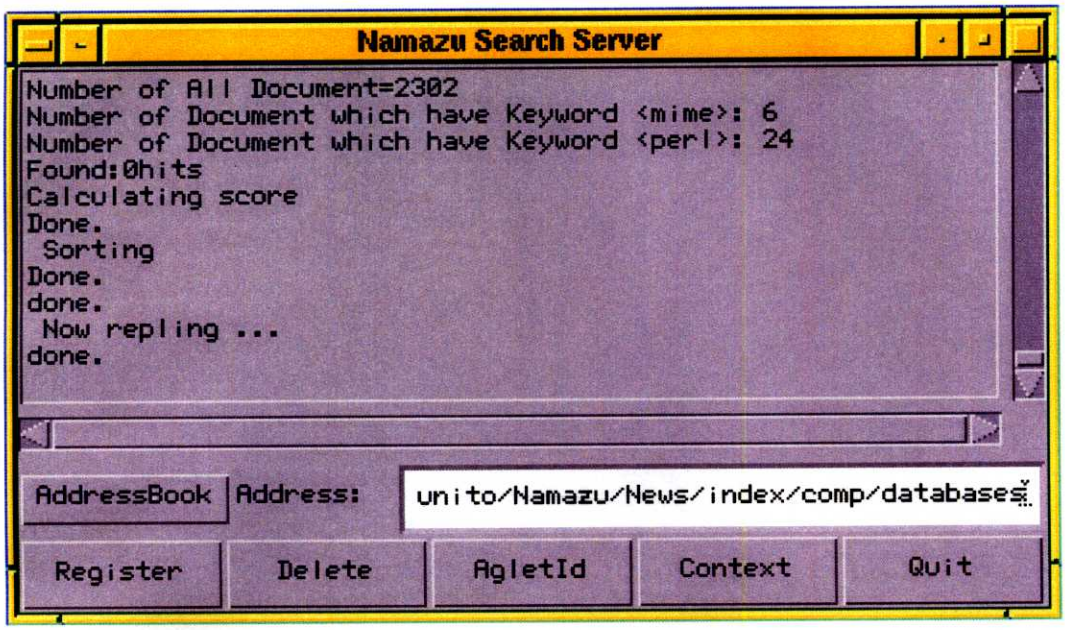


Figure 7.7: Search Server

現在登録されている Search Server のリストを返す。“put” メッセージによって Search Server を登録することができる。重複チェックを行っているため重複登録は防止できる。“delete” メッセージによって登録を削除することができる。“clear” は保守用のメッセージで、リストを空にすることができる。何らかの原因で本来存在しない Search Server が残ってしまった場合などに使用する。

- Search Server
  メッセージの種類としては “query” “recalc” を受け付ける。query メッセージは SC からであっても他の SS からであっても構わない。したがって SS が他の SS に検索要求を行うことで再帰検索を行うことができる。query と共に送られた keywords を含むコンテンツを検索する。現在は AND 検索のみが実装されているが、他の検索方式も拡張可能である。query message に対する reply という形で検索結果を返す。recalc は協調検索プロトコルにおいてのスコアの再計算要求のメッセージである。

  本実装では検索部分も All Java 化してあるため、Namazu の検索クライアントである namazu プログラムは使用していない。

- Search Client
  “query” メッセージを受け付ける。本実装においては UI 起動時に自動的に SC を子エージェントとして起動するようにしている。これは UI が SC の場所、ID を取得する必要が

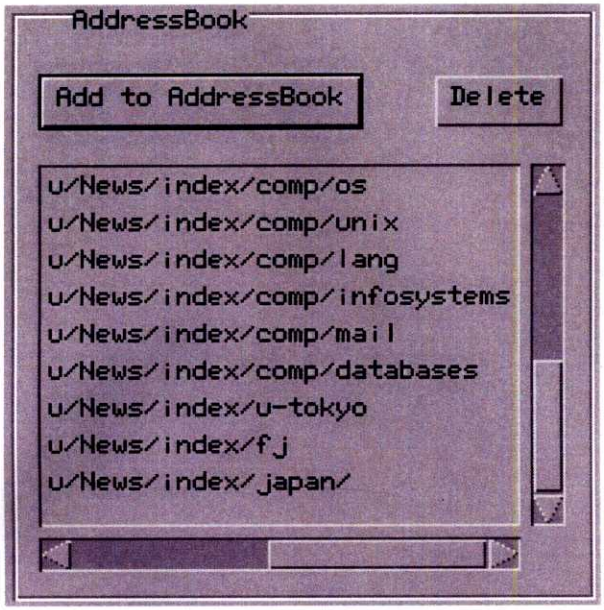


Figure 7.8: Address List of Search Server

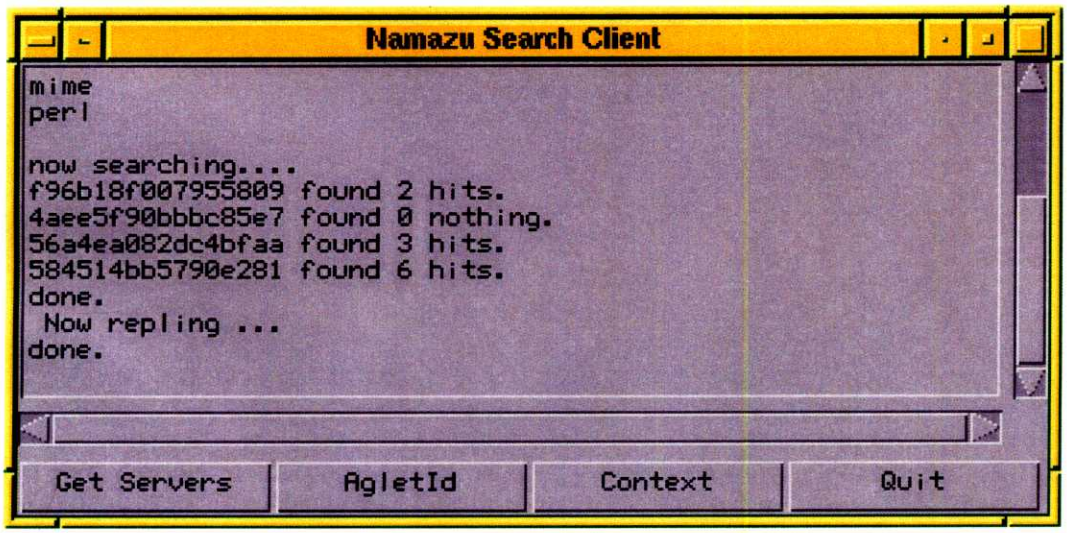


Figure 7.9: Search Client

ないようにするためである。実装形態によって PDA のような非力なマシン上で UI が動作する場合には SC の場所を UI に教えるような仕組みにしておけば、親エージェント以外の

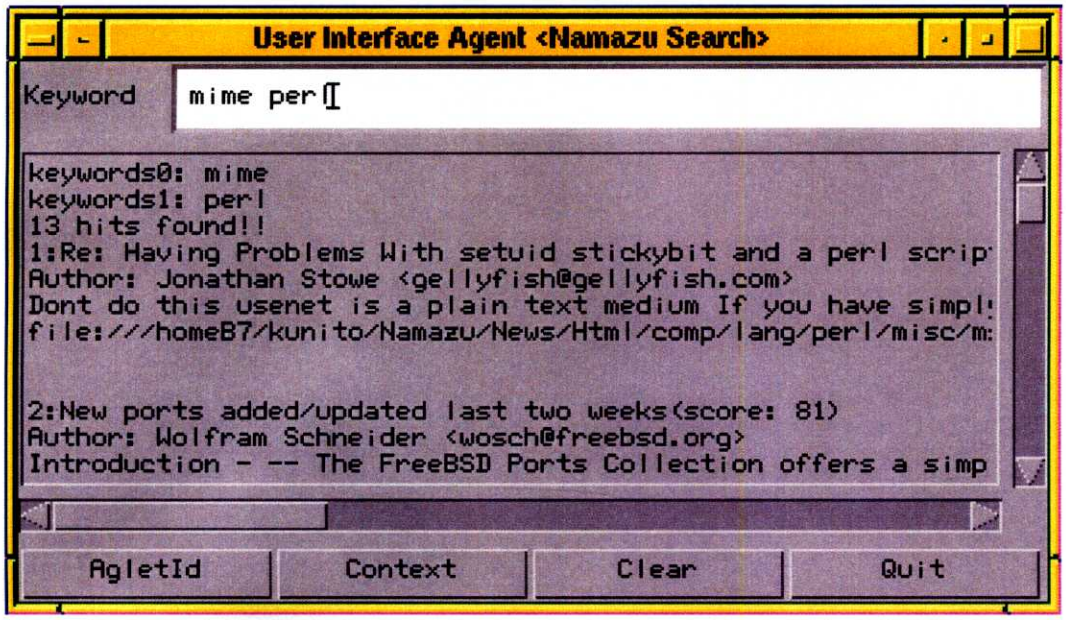


Figure 7.10: User Agent

UI からの検索要求を受け付けることもできよう。

本実装では Search Client は Serach Server からヒットしたドキュメント全てと、Tf·Idf によるスコア計算に必要なそれぞれのサーバ上の全文書数ならびにキーワードを含む文書数を取得している。そのため Search Client 上で全てのサーバのドキュメントを考慮した正しいスコア計算を行うことができる。

7.7 おわりに

携帯情報端末の普及によりモバイルコンピューティングが現実のものとなっており、個人が持ち歩く携帯端末の中に蓄えられる情報も増加し、検索システムが求められるような状況が今後ますます増えるであろう。大きな検索システムばかりではなく、Namazu のような手軽な検索システムが今後ますます活躍することが予想される。

本論文では個人が持っているような小さなデータベースを、個人的に使うだけでなく他の小さなデータベースと連携させていこうという方向を目指している。なかでも、連携においてユーザへのレスポンスを所定の時間内に押さえるために、再帰検索の打ち切り法を検討した。

携帯情報端末上の検索システムの連携はこれからのモバイルエージェントのアプリケーションとして期待できよう。今後課題を解決しながら実装を進めていく予定である。

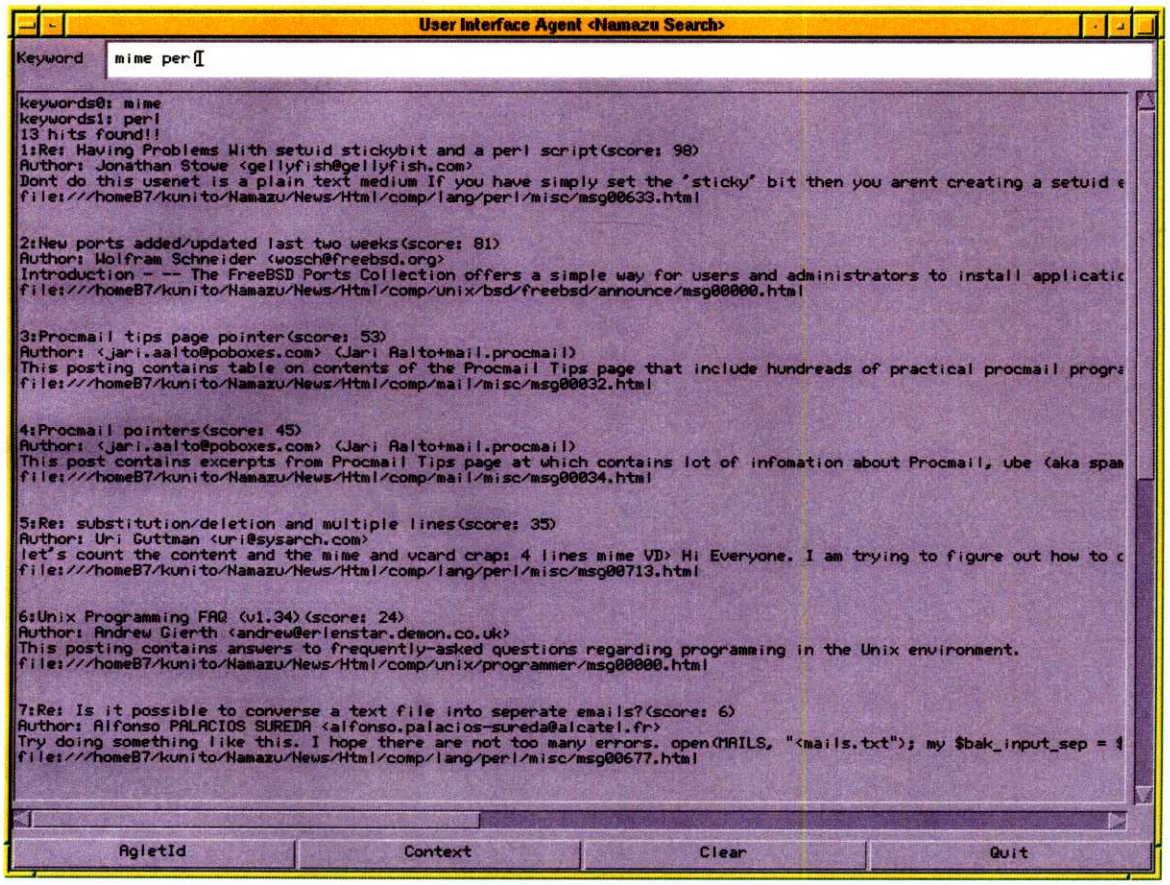


Figure 7.11: Example of Result

Bibliography

[1] Alta Vista. http://www.altavista.digital.com/.

[2] 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.

[3] Goro Kunito, Kiyoharu Aizawa, and Mitsutoshi Hatori. Tracking agent : A new way of communication in a multi-agent environment. In *The 6th IEEE International Conf. on Universal Personal Communications (ICUPC'97), San Diego, California, U.S.A.*, Oct 1997.

[4] Robert H'obbes' Zakon. Hobbes' internet timeline v3.3. http://www.isoc.org/zakon/Internet/History/HIT.html.

[5] Yahoo! http://www.yahoo.com/.

[6] 國頭吾郎, 相澤清晴, 羽鳥光俊. エージェント間通信のための tracking agent の提案とそ の検討. 電子情報通信学会総合大会, pp. B–7–242. 電子情報通信学会, Mar 1997.

[7] 吉岡恒夫. 代理サーバを利用した検索システム. http://infonet.aist-nara.ac.jp/~tsuneo-y/proxysearch/.

[8] 高林哲. 全文検索システム Namazu. http://www.ring.gr.jp/openlab/namazu/index-j.html.

[9] 馬場肇. 日本語全文検索システムの構築と活用. ソフトバンク, Sep. 1998.

Chapter 8

結論

本論文は、将来のマルチメディア統合ネットワークにおけるエージェントシステムの実現に向けて、エージェント間通信の新しい枠組みおよびそのアプリケーションを論じたものである。ここでは本論文の主たる成果についてまとめ、ついで今後のエージェント技術の展望について述べ、全体を統括する。

8.1 本論文の主たる成果

本論文では、将来のマルチメディアサービスにおける各ユーザの「個」を実現することを最終目標とし、モバイルエージェント間の通信支援の方式やエージェントの階層化といったエージェント技術の在り方を探求してきた。具体的には、

- 複数モバイルエージェント間の通信を支援するためのトラッキングエージェント
- 階層化マルチエージェントシステム
- モバイルエージェントを用いた複数の検索システムの連携

に関する検討を行った。以下、本論文の構成に沿いながら各項目を要約する。

本論文ではエージェントを扱っている。エージェントと呼ばれている分野は実に多岐にわたっているため、第2章ではエージェントに関する基礎知識を提供すべく、用途・背景を取り上げ概観している。多様なエージェント技術の中で本論文ではモバイルエージェントに着目した。現在のモバイルエージェントのアプリケーションとしては単独のモバイルエージェントが多数の仕事を行ってくる例が多いが、将来は複数のモバイルエージェントが協調しつつ仕事を行うことが予想される。第3章では、そのような場合にモバイルエージェント同士がシームレスに通信をしながら仕事を行えるような新しい通信の枠組みを提案した。このトラッキングエージェントを用いると、モバイルエージェントの移動先に偏りのある場合に特にメッセージの遅延やモバイルエージェントの移動にかかる時間が小さくなることが計算機シミュレーションによって明らかになった。第4章では、まず実装におけるエージェント言語を概観した上で、AgentTcl という言語を用いたトラッキングエージェントの実装について述べた。第5章では一対多のエージェント間通

信のための階層化マルチエージェントシステムを提案した。エージェントを階層的に配置することで中間処理なども可能となり、中心的なエージェントへの負荷が軽減され、またエージェントの数やトラヒックに応じて動的に階層構造を変化させることによってメッセージの遅延などを改善できることが示された。

第 6 章、第 7 章では WEB の検索システムとエージェントの関係に着目した。第 6 章では、Web ブラウズによるトラヒック軽減のために近年普及している Proxy Cache のトラヒックを解析し、単に Cache としてだけではなく、この Cache したデータを検索システム構築のためのコンテンツとして利用可能の量であることを示した。また、第 7 章でモバイルエージェントを用いることで複数の小規模検索システムが連携することができることを示した。

8.2　エージェント技術の今後の展望

エージェントの存在意義として本論文では「モバイル」な部分に最も注目している。従来のクライアントサーバ型の通信に対して、エージェント型の通信はプログラム自体が移動することによって通信遅延の影響を受けにくいという特徴がある。また、エージェント指向はオブジェクト指向からさらに一歩進んだ新しいパラダイムであると言える。

本論文ではいくつかの切り口でエージェントを整理した。もちろんそれぞれの分類は排他的なものではなく、複数の分類項目に跨ったエージェントが存在することは十分にありうる。しかしながら、全ての項目を兼ね備えたエージェントは、実装・実現することは困難であると思われる。その点でエージェントが絶対的にあらゆる全ての既存技術よりも全ての面で優れていると言うことは困難である。しかしながら、エージェント技術が既存技術と競合するものであることは間違いないであろう。したがって、何がなんでもエージェントを使う、というよりも、有望な選択肢の一つとしてエージェント技術があると考えることが有効であろう。

様々なネットワークサービスの提供される将来のネットワークにおいては、様々なサービスに対応するためにネットワーク自身も動的にサービスとの協調、他のネットワークとの調整等がこれまで以上に必要になってくると考えられる。これまでも Router などは他のルータとの Routing Table の交換や帯域の予約などの調整をしてきているが、今後はもっと動的で複雑な制御が必要となって知識が必要となると思われる。そのような知的で動的なネットワークの実現の要素技術の一つとしてエージェントの活躍が期待される。

■■■ 発表論文 ■■■

■ 学会誌論文

[1] 國頭吾郎, 相澤清晴, 羽鳥光俊. 協調エージェント間通信のためのトラッキングエージェントの提案とその検討. 電子情報通信学会論文誌 BI, Nov. 1998.

[2] 國頭吾郎, 吉川典史, 相澤清晴. 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. 情報処理学会論文誌『マルチメディア通信プロトコル』特集, Vol. 41, No. 2, Feb. 2000. 掲載予定.

■ 国際会議

[3] Goro Kunito, Kiyoharu Aizawa, and Mitsutoshi Hatori. Tracking agent : A new way of communication in a multi-agent environment. In *The 6th IEEE International Conf. on Universal Personal Communications (ICUPC'97), San Diego, California, U.S.A.*, Oct 1997.

[4] Goro KUNITO, Kiyoharu AIZAWA, and Mitsutoshi HATORI. An Application of Mobile Agent for WWW Search System. In *The '99 International Symposium on Teletraffic Modeling of Mobile Multimedia Communication Systems.* Telecommunications Advancement Organization of Japan(TAO), and Korea Institute of Communication Sciences(KICS), Mar 1999.

[5] Goro KUNITO and Kiyoharu AIZAWA. A Mobile Agent Based WWW Search System. In *The '99 International Symposium on Communications*, Nov. 1999.

■ 学会研究会

[6] 國頭吾郎, 相澤清晴, 羽鳥光俊. 協調エージェントのための通信に関する検討. 信学技報, Vol. MVE96-47, pp. 11–16, Nov 1996.

[7] 國頭吾郎, 相澤清晴, 羽鳥光俊. 協調エージェントのための通信に関する検討. 信学技報 MVE96-47, 電子情報通信学会, Nov 1996.

[8] 國頭吾郎, 相澤清晴, 羽鳥光俊. 複数エージェント間の通信のための tracking agent に関する検討. 第 19 回情報理論とその応用シンポジウム (SITA96), pp. 417–420, Dec 1996.

[9] 國頭吾郎, 奥村恭弘, 相澤清晴, 羽鳥光俊. 協調エージェント間通信のための tracking agent に関する 検討. 信学技報 IN97-20,CS97-1,MVE97-1, 電子情報通信学会, Apr 1997.

[10] 國頭吾郎, 相澤清晴, 羽鳥光俊. 協調エージェント間通信のための Tracking Agent の提案 とその検討. マルチメディア、分散、協調とモーバイル (DiCoMo) ワーク ショップ, pp. 563–568. 情報処理学会, Jul. 1997.

[11] 國頭吾郎, 相澤清晴, 羽鳥光俊. Tracking Agent を用いたエージェント間通信での移動端末

を考慮したMobile Agentの 移動制御に関する検討. マルチメディア、分散、 協調とモーバイル (DiCoMo'98) シンポジウム, pp. 153–158. 情報処理学会, Jul. 1998.

[12] 吉川典史, 國頭吾郎, 相澤清晴, 羽鳥光俊. 階層化エージェント. マルチメディア、分散、 協調とモーバイル (DiCoMo'98) シンポジウム, pp. 159–164. 情報処理学会, Jul. 1998.

[13] 國頭吾郎, 相澤清晴, 羽鳥光俊. 協調エージェント間通信のためのト tracking agent の提案とその検討, Nov 1999. 情報処理学会 MBL Symposium 優秀論文招待講演.

[14] 國頭吾郎, 相澤清晴, 羽鳥光俊. モバイルエージェントを用いた WWW 検索システムに関する一検討. 信学技報 MVE98-93, 電子情報通信学会, Feb 1999.

[15] 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, Feb 1999.

[16] 國頭吾郎, 相澤清晴. モバイルエージェントを用いたWWW 検索システムの協調に 関する検討. マルチメディア、分散、 協調とモーバイル (DiCoMo'99) シンポジウム, pp. 145–150. 情報処理学会, Jun.-Jul 1999.

■ 国内大会

[17] 國頭吾郎, 相澤清晴, 羽鳥光俊. 統合メディア通信におけるパーソナルエージェントに関す る考察. 電子情報通信学会ソサエティ大会, pp. B–818, Sep. 1996.

[18] 國頭吾郎, 相澤清晴, 羽鳥光俊. エージェント間通信のための tracking agent の提案とそ の検討. 電子情報通信学会総合大会, pp. B–7–242. 電子情報通信学会, Mar 1997.

[19] 奥村恭弘, 國頭吾郎, 相澤清晴, 羽鳥光俊. エージェント間通信のためのトラッキングエージェントの実装. 電子情報通信学会全国大会, Mar 1997.

[20] 國頭吾郎, 相澤清晴, 羽鳥光俊. Tracking Agent の応用に関する一検討 — 協調モーバイルエージェントによる電子商取り引きへの応用例 —. 第 55 回 (平成 9 年後期) 全国大会, No. 2Q-3 in 3, pp. 3–176. 情報処理学会, Sep. 1997.

[21] 中川智尋, 國頭吾郎, 相澤清晴, 羽鳥光俊. モバイルエージェント間の通信の為の Tracking Agent の Java による実装. 電子情報通信学会全国大会, pp. B–7–46, Mar 1998.

[22] 國頭吾郎, 相澤清晴, 羽鳥光俊. Tracking Agent を用いたエージェント間通信における Mobile Agent の移動に関する検討. 電子情報通信学会全国大会, pp. B–7–47, Mar 1998.

[23] 國頭吾郎, 相澤清晴, 羽鳥光俊. トラッキングエージェントを用いた複数エージェント間通信の WWW 検索サービスへの応用に関する検討. 第 57 回 (平成 10 年後期) 全国大会, No. 3F-2 in 3, pp. 3–346. 情報処理学会, Oct. 1998.

[24] 吉川典史, 國頭吾郎, 相澤清晴, 羽鳥光俊. 階層的マルチエージェントシステムに関する一検討. 第 57 回 (平成 10 年後期) 全国大会, No. 3F-4 in 3, pp. 3–350. 情報処理学会, Oct. 1998.

[25] 國頭吾郎, 相澤清晴, 羽鳥光俊. WWW 検索サービスへの協調モバイルエージェントの 応用に関する一検討. 電子情報通信学会 1999 年総合大会, pp. B–7–25, Mar 1999.

[26] 吉川典史, 國頭吾郎, 相澤清晴, 羽鳥光俊. 階層的マルチエージェントシステムの動的再構成.

電子情報通信学会 1999 年総合大会, pp. B–7–62, Mar 1999.

[27] 國頭吾郎, 相澤清晴. モバイルエージェントを用いた検索システムの連携における一検討. 第 59 回 (平成 11 年後期) 全国大会, No. 3S-8 in 3. 情報処理学会, Oct. 1999.

[28] 國頭吾郎, 相澤清晴. 協調検索システムにおける継続検索に関する検討. 電子情報通信学会 2000 年総合大会, Mar 2000 (発表予定).

■■■　謝　辞　■■■

終りに臨み、本研究を通じて終始御指導を賜った 相澤　清晴　助教授 ならびに 羽鳥　光俊　教授（現在、名誉教授）に深甚なる謝意を表します。

その他、田中　崇　助手、塚本　憲男　技官をはじめとする羽鳥・相澤研究室の諸先輩・後輩の方々には、研究活動を進めるにあたって数々の便宜を図って頂きました。ここに深く感謝致します。最後に、これまでの研究生活を物心両面から支援してくれた両親と友人に深く感謝の意を表します。

1999 年 12 月 17 日