特集：情報プラットフォーム

研究論文

モバイルエージェント技術と研究動向

Mobile Agent Technology

佐藤 一郎
国立情報学研究所
Ichiro SATOH
National Institute of Informatics

要旨

モバイルエージェントはコンピュータ間を移動可能な能動的なプログラムであり，分散処理における通信遅延及び回数の削減，対故障性の向上などの数多くの利点があり，次世代の分散処理システムとして注目集めている．本稿ではこのモバイルエージェントの動作原理として，エージェントのコンピュータ間移動機構，エージェントプログラムの実行方法，セキュリティなどについて概説する．そして，実用性の見地から代表的な応用事例をいくつか紹介するとともに，既存のモバイルエージェントシステムについてその代表的なものを取り上げ，その特徴をまとめる．最後に今後の研究動向について概観する．

ABSTRACT

Mobile Agents are autonomous programs that can travel from computer to computer under their own control. They can provide a convenient, efficient, and robust framework for implementing distributed computing systems, including mobile computing systems. Mobile agent technology is promoted as an emerging technology that makes it much easier to design, implement and maintain distributed systems. The paper explains how existing mobile agent systems migrate agents between computers and execute agent programs, and then present practical applications of the technology. Also, it surveys several existing mobile agent systems and briefly describes some future trends of the technology.

[キーワード]

モバイルエージェント，分散システム，ネットワーク，ミドルウェア

[Keywords]

Mobile Agent, Distributed System, Network, Middleware

1 はじめに

モバイルエージェントは，コンピュータ間の自律的移動能力をもつプログラムである．その移動においては，プログラムのコードだけでなく，それの実行状態，つまりプログラム内の変数内容なども移動先に転送される．この結果，移動先コンピュータでは移動前の実行状態から処理を継続することができる．なお，モバイルエージェントは他のエージェント技術とは異なり，インテリジェンスはもたないことが多いが，能動的に実行され，移動先もそれ自身により選択し，移動することができる．

モバイルエージェントに関する研究は，90 年代の中頃に登場した商業用モバイルエージェントシステムである Telescript[1] 以降本格化する．そして，1997 年頃からは Java 言語を利用したモバイルエージェントシステムが数多く研究開発されている．また，モバイルエージェントの応用分野も，従来の遠隔情報収集や負荷分散だけでなく，通信ネットワーク管理やワークフローシステムなどにも利用され，広がりをみせている．

本稿では，モバイルエージェントの概説とともに，応用分野と研究動向を紹介するものである．次章ではモバイルエージェントの特性を概説し，3 章ではモバイ

ルエージェントの実現方法について解説する．4 章では具体的な応用事例を示し，続く 5 章では既存のモバイルエージェントシステムを紹介する．6 章では研究動向を，7 章では標準化活動を紹介する．最後の 8 章でまとめを述べる．

2　モバイルエージェント

モバイルエージェントは次世代の分散システムの中核技術として注目を集めている．本章では分散システムの観点からモバイルエージェントの特徴を列挙していく．

遠隔実行

コンピュータ間通信の遅延時間は大きく，分散システムの性能上のボトルネックとなっている．ここで，モバイルエージェントを利用することにより通信コストを低減することができる．これは，エージェントとして実現された分散処理プログラムを通信相手のコンピュータに移動・実行させ，コンピュータ間で行われていた通信を一つのコンピュータ内のプログラム間通信に局所化するものである．この結果，コンピュータ間通信はエージェントの移動時だけに最小化され，移動後の処理は移動先コンピュータ内の処理となる．また，セキュリティ上許されるのであれば，モバイルエージェントは，移動先の計算リソースをそれ自身のプログラムにより直接制御・アクセスすることもでき，効率化とともに柔軟性も向上する．

非同期実行

移動後のモバイルエージェントは，その移動先のコンピュータ内で処理されるため，移動前のコンピュータと通信を行う必要はない．これはモバイルエージェントが空間的にも時間的にも非同期に実行されることを意味しており，移動端末や無線通信のように一時的な通信切断が頻発するシステムや，電話回線のように経済的な理由から常時接続は望めないシステムにおいて有用となる．

分散プログラミングの簡易化

従来の分散システム，例えばクライアント・サーバモデルではクライアントとサーバの２種類のプログラムが必要であり，遠隔手続き呼び出しでもスタブとスケルトンが別途に必要となる．また，それらを遠隔コンピュータに事前にインストールしておかなければならない．一方，モバイルエージェントの場合，エージェントの移動や実行管理を行うランタイムシステムを各コンピュータに稼働しておけば，エージェント移動によりプログラムの事前配置が行える．また，そのプログラム自体はコンピュータ間通信を行う必要はなく，これもエージェント移動により代用することができる．従って，通信などの外部開放性を持たない閉じたプログラムにより，分散処理が記述できることになり，用意するプログラムも一つでよい．さらに通信失敗などへの例外処理も簡単化される．

補足

モバイルエージェント以前にも，プログラムのコンピュータ間移動技術は数多くあった．例えば，Java Applet や Postscript などは遠隔コンピュータの実行が前提となるが，実行状態は移動対象とならないため，常に初期状態から処理を再開しなければならない．逆に，モバイルエージェントと同様に実行状態の移動性をもつ技術として，プロセス移動やオブジェクト移動などがある．モバイルエージェントの移動技術はそれらの延長線上にあるが，プログラム粒度的には両技術の中間に位置するものとなる．

3　モバイルエージェントの実現

本章ではモバイルエージェントの動作原理やその実現方法について概説する．モバイルエージェントの記述は独自言語によるものもあるが，ソフトウェア資産を再利用する立場から Java 言語や Tcl 言語などの既存プログラミング言語で記述されるものが多い．そして，移動性などのモバイルエージェント特有の機能は，言語拡張または API ライブラリにより提供されている．

なお，エージェントの実行には，ランタイムシステムやエージェントホストなどと呼ばれるソフトウェアが，コンピュータ上に予め動作している必要がある．このランタイムシステムはエージェントの実行管理や移動などを実現し，この他，エージェントの永続化 (エージェントを必要なときに活性化できるようにファイルに保存できるようににする) やエージェント間通信機能なども提供する．

3.1　エージェントの実行

モバイルエージェントは，相違なハードウェアや OS をもつコンピュータに移動することがある．このため，

エージェントは多様な計算環境で実行できることが必要であり，その方法には大きく分けて以下の３つがある．

(1) インタプリタまたは仮想機械によりエージェントを解釈・実行する方法．これはエージェントをソースプログラムまたは中間言語などのコンピュータに独立な形式で表記するとともに，各コンピュータに用意したインタプリタまたは仮想機械を通じて解釈・実行するものである．多様な計算環境に対応できるが，実行コストが大きくなる．

(2) ネイティブコードに変換・実行する方法．これはエージェントをハードウェアや OS に独立な形式で記述しておき，実行開始時や到着時にまずネイティブコードに変換する．そして変換したネイティブコードを通じてエージェントを実行する方法である．(1) よりは効率化される．ただし，再び移動するとき，ネイティブコードによる実行状態を独立形式の表記に変換することになるが，ネイティブコードの実行状態は一般に複雑となり，変換できないことがある．

(3) 各コンピュータのネイティブコードを保持・実行する方法．エージェントは移動先となるすべてのコンピュータのネイティブコード形式によるプログラムを重複保持する．そして，移動時にはその移動先に適したネイティブコードを選択・実行する．エージェント実行は効率化されるが，移動時のデータ量が大きくなるという問題がある．

補足

既存のモバイルエージェントでは，移植性や後述のセキュリティの観点から，(1) の方法が主流になっている．実行性能についても，実行時に仮想機械コードをネイティブコードに変換しながら実行する技術 (Just-In-Time コンパイラなど) が発展し，ネイティブコード実行との性能差は小さくなっている．また，ランタイムシステム自体も仮想機械上で動作させて，移植性を高めていることが多い．

3.2 エージェントの移動

エージェントのコンピュータ間移動について，その過程を概説する．モバイルエージェントを実現するシステムは数多くあるが，エージェントのコンピュータ間移動は，遠隔手続き呼び出しの引数受け渡しと同様

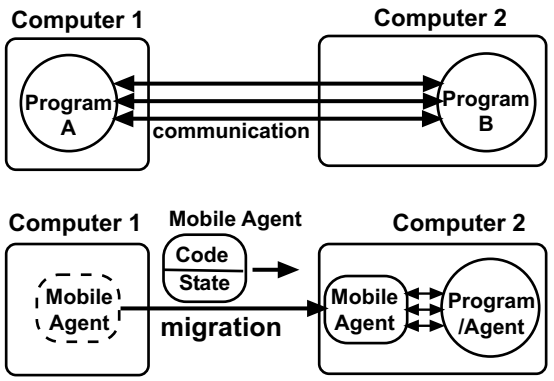

図 1: エージェントのコンピュータ移動

の方法で実現している．その概要を図 1 に示す．

(1) 移動対象となるエージェントの実行を一時中断する．

(2) エージェントの実行状態をデータ化する (直列化[1])．そのデータ化した実行状態とプログラムコードを転送可能なデータ形式に符号化する．

(3) データ通信プロトコルを利用して，符号化したデータを移動先のコンピュータに送信する．

(4) 受信したデータを復号化し，再びエージェントに変換する．

(5) エージェントの実行を再開する．

なお，エージェント移動に用いるデータ通信プロトコルは多様であるが，Java 言語によるシステムでは Java 言語の RMI(Remote Method Invocation) 機能を利用しているものが多い，ただし，この他にも電子メールや独自の通信プロトコルを利用するものがある．

3.2.1 実行状態の移動

モバイルエージェントの移動は，コードだけでなく実行状態も転送されることが特徴となる．ただし，この実行状態の範囲についてはシステムにより相違することから注意が必要である．

方式 A: Code Migration コードだけを転送する．このため，移動先では初期状態から実行を開始することになる．この例には Java Applet，JavaScript, PostScript などがある．

[1]遠隔手続き呼び出し (RPC) などでは直列化 (Serialization) より，整列化 (Marshalling) と呼ぶことが多い．

方式 B: Weak Migration プログラムコードに加えて，インスタンス変数などのヒープ領域内の情報を実行状態として転送する．なお，Java 言語の JDK 1.1 以上で提供される直列化機構 (Serialization) は本方式に基づいており，その結果，Java 言語によるモバイルエージェントシステムの多くが本方式となっている．

方式 C: Strong Migration 方式 B の範囲に加えて，実行中メソッドや関数のローカル変数などのスタック領域内の情報，さらにプログラムカウンタも転送する．ただし，多重化された実行スレッドについては移動対象にならないことが多い．

多くのモバイルエージェントは方式 B または C を用いているが，どちらをとるかについては議論が分かれている．方式 B はヒープ領域内の情報に限定しているため移動コストが小さくなる．しかし，プログラムカウンタやローカル変数は対象外となる．従って，移動直前の実行箇所から再開することは難しく，移動先ではエージェント内のメソッドや関数を明示的に呼び出す機構が必要となる．一方，方式 C では，あるメソッド内を実行している最中に移動することになっても，移動先では移動直前の実行箇所からそのまま処理を再開でき，移動性に備えた特別な記述が不要となる．

しかし，実際的アプリケーションの記述では，方式 B で十分であるという報告が多い (例えば[2,3])．これは方式 C を採用しても，移動前後にファイルやウィンドウなどの各種計算リソースの解放・獲得を行う例外的処理が必要となり，移動直前の実行箇所からそのまま再開できるとは限らないからである．また，モバイルエージェントはそれ自身が主体となって移動することから，移動タイミングも制御でき，方式 B でもローカル変数などをヒープ領域に待避することができる．この他，Java 言語などの仮想機械方式では，その実行性能が Just-In-Time コンパイラなどの最適化技術に依存していることが多いが，方式 C はこれらの最適化技術との整合性が悪い．

3.2.2 プログラムコードの移動

プログラムコードに関しても複数の移動方式が提案されている．(1) 予め移動先にプログラムコードを配布しておく方式．エージェント移動に際して，コード転送が不要となることから，移動が効率化する．(2) エージェント実行中に必要に応じてコードをダウンロードする方式．ただし，Java 言語のようにクラスを動的に束縛する場合には，いつコードをダウンロードするかが不明なため，エージェント移動後もネットワークが接続されている必要がある．(3) 実行状態とともに必要なコードを一括移動する方式．通信切断に対応できるようになるが，移動データ量が大きくなる．

3.3 セキュリティ

モバイルエージェントは他のコンピュータに移動し，処理を行うことから，移動先コンピュータの各種リソースを不正にアクセスする可能性がある．このため，セキュリティ・保護機構が必要不可欠となる．なお，仮想機械またはインタプリタによりエージェントを実行するときは，仮想機械またはインタプリタに制限を加えて，ファイルやネットワークに関する操作を禁止する方法や (例えば[4])，特定ライブラリコードのロードを制限する方法が利用されている．

一方，ネイティブコードによる実行では，不正なコードが含まれるとシステム全体に影響を与えることから，事前検証は必須となる．ただし，メモリアクセスの正当性は実行時にしか判定できないことが多く，強いデータ型をもつプログラミング言語により記述することが望まれる．ファイルなどの各種リソースへのアクセスについては，リソースに関わるシステムコールや API を横取りし，違法な操作を禁止する方法がある．この他，プログラム動作内容に関する証明を添付する方法 (例えば Proof Carrying Code[5]) や，プログラムを変換して，メモリやリソースに関わるコードの前後に正当性をチェックするコードを挿入し，その変換済みプログラムを実行する方法が提案されている (例えば Software Fault Isolation[6])．

4 モバイルエージェントの応用

モバイルエージェントにはキラーアプリケーションがないといわれている．しかし，モバイルエージェントを利用することにより，高機能化や開発効率の低減を図ることのできるアプリケーションは数多く存在する．ここでは，筆者らがすでに開発・稼働している応用事例を中心に紹介していく[2]．

(1) 遠隔検索 モバイルエージェントを移動させ，移動先においてデータベースの検索やセンサー情報の収集

[2]本節で紹介した応用事例の多くは，AgentSpace システムのサンプルプログラムとしてホームページ `http://islab.is.ocha.ac.jp` よりダウンロード可能である．

を行い，その結果を移動元に持ち帰らせるものである．このとき，モバイルエージェントは，移動先においてそれ自身のプログラムによりフィルタリングや圧縮ができることから，通信量を低減することができる．また，検索結果に基づいて別のコンピュータに移動し，検索を続けることもできる．

(2) 分散リアルタイム処理　分散リアルタイムシステムの難しさの一つは，通信遅延の変動がリアルタイム制約に影響することである．ただし，モバイルエージェントを利用すると，エージェント移動によりコンピュータ間通信を一つのコンピュータ内のプログラム間通信に局所化することができる．コンピュータ内のプログラム通信は通信遅延が小さいだけでなく，その遅延時間の変動も小さくなることから，リアルタイム制約の充足判定が容易になる．

(3) 負荷分散・耐故障性　モバイルエージェントの移動性により，所定時間後にシャットダウンするコンピュータや計算負荷が高いコンピュータから，他のコンピュータに待避させ，その待避先でそのまま処理を継続させることができる．また，複数コンピュータに明示的に移動させることにより，処理の並列化も可能となる．

(4) ネットワークの巡回保守・管理　モバイルエージェントを巡回させ，ネットワーク上の各種機器の監視・制御や，ソフトウェアの自動インストールを行うことができる．特に，モバイルエージェントはその実行状態とともに移動できることから，各種設定を済ませたソフトウェアを配布・稼働もできることから，システム管理コストの低減にも有効となる．

(5) ネットワークセキュリティ　非公開情報や有料情報をモバイルエージェントに内包・暗号化して移動させることにより，転送中の盗聴や改ざんを防ぐことができる．特に，暗号解読プログラムもエージェントに内包できるため，移動先に解読プログラムを用意する必要はなく，任意の暗号形式を用いることができるようになる．

(6) パーソナルアシスタンス　既存のモバイルコンピューティングでは，ユーザ自身がコンピュータを持ち歩くことが前提となるが，モバイルエージェントを利用することにより，手近にあるコンピュータに自分のパーソナルアシスタントとなるエージェントや，アプリケーションプログラムを含むデスクトップ環境を移動させ，利用することができる．従って，ユーザはコンピュータの持ち歩かなくてよくなる．著者らはモバイルエージェントによる移動ユーザへの支援システムの開発を進めている[7,8]．

(7) ワークフロー・メールシステム　グループウェアなどにおいて，一つのワークフローやメールをモバイルエージェントとして実現することにより，処理内容や担当者の不在などの状況変化に応じて処理経路や宛先を変更できるようになる．また，ワークフローやメールの内容を編集するプログラムも同時に移動できるため，任意のデータ形式を扱えるようになる．

(8) アクティブネットワーク　新しい通信プロトコルを普及させることは簡単ではない．しかし，通信プロトコルの処理プログラムをモバイルエージェントを通じて移動させることにより，そのプロトコルに対応していないコンピュータにインストール・稼働させることができる．これにより，通信プロトコルの標準化作業自体を不要にすることも可能である．

(9) ソフトウェアコンポーネント　モバイルエージェントが移動可能なソフトウェアコンポーネントとして導入することが注目されている．これによりコンポーネントをシステムに動的インストールしたり，不要なものを取り除くことができるため，拡張可能または発展的ソフトウェア構成の基礎となりうる．現在，モバイルエージェントによる複合ドキュメント（多様なコンポーネントからなる文書データ）などの研究がある[9]．

5　既存のモバイルエージェントシステム

モバイルエージェントを実現する既存ソフトウェアをいくつか紹介する．

5.1　Telescript

Telescript[1] は世界初の商用モバイルエージェントシステムであり，エージェントをネットワーク上を移動させて情報を収集させたり，電子商取引をさせることを意図して開発された．Telescript のモバイルエージェントの他に次の 5 つの概念，プレース，トラベル，ミーティング，オーソリティ，パーミットにより特徴づけられる．

- **プレース**は各コンピュータ上に 1 つ以上配置される移動性をもたないエージェントである．プレー

スはその内部に 0 個以上のプレースまたはモバイルエージェントを含むことができ，その内部エージェントに対してそれ自身のリソースやサービスを提供する場となる．

- **トラベル**とはエージェントがあるプレースから別のプレースに移動することである．このとき，それらのプレースは同一または相違なコンピュータ上にあってもよい．エージェントが移動するときははコマンド `go()` を実行するが，その引数としてチケットと呼ぶデータが必要となる．チケットには移動先の他に，移動方法，移動するまでの時間制限，後述するパーミットなどが記載されており，移動の諸条件を定義する．
- **ミーティング**は同じプレースに存在するエージェント同士の通信を表す．エージェントはコマンド `meet()` を実行し，同じプレース上の他のエージェントに対して通信要求を行う．引数には通信相手や通信開始時間などを指定する．通信が開始されると互いのメソッドを呼び出すことができる．コマンド `part()` を実行することにより通信を終了する．
- **オーソリティ**はエージェントやプレースの識別子であり，システムによって割り当てられるユニークな番号の他に，エージェントやプレースの所有者が実社会で属する組織や名前に対応した番号をもつ．Telescript は電子商取引に応用することが前提となっているが，このオーソリティによりエージェントが行った商取引とその所有者を正確に対応させることができる．
- **パーミット**はエージェントやプレースに与えられ，それらの能力を表し，特定命令の実行強化の他，エージェントの生存時間，CPU 消費量，エージェントのバイトサイズの上限などを定める．エージェントがパーミットの制限量を超えると破棄される．ただし，エージェントは制限量に達する前に通知を受けることができ，残ったパーミットで例外処理を実行することができる．このパーミットを導入したことにより，特定のエージェントが不正アクセスをしたり，計算機の資源を不当消費することを防ぐことができる．

エージェント及びプレースは，Telescript と呼ばれる独自のオブジェクト指向言語の記述される（下記にプログラム例を示す）．そして，バイトコードにコンパイル

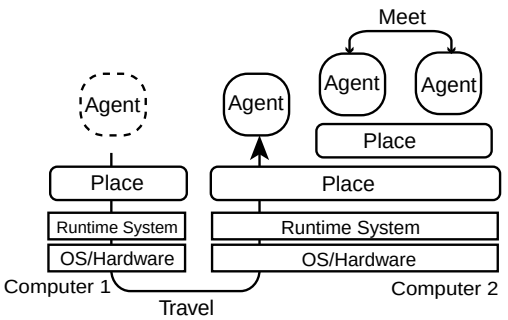

図 2: Telescript エージェント

され，仮想機械上で実行される．Telescript のシステムはこの仮想機械に加えて，エージェントのコンピュータ間移動を実現する通信機構と，外部プログラムの呼び出し機構からなる．エージェントの移動には TCP/IP，電子メール，電話会社固有の通信プロトコルが利用でき，その際にはコードだけでなく，スタックやプログラムカウンタも移動し，移動先では `go()` コマンドの実行直後からそのまま処理を再開する．

Telescript は商業的には成功しなかったが，その後のモバイルエージェントに大きな影響を与えた．また，General Magic 社は Telescript を Java 言語により実現したシステムである Odyssey[10]を公開している．

```
Shopper: class (Agent, EventProcess) = (
  public
    see initialize
    see meeting
  private
    see goHome
  property
    ....
);
```

リスト 1: クラス宣言例 (一部)

```
initialize: op (
  desiredProduct: owned String;
  desiredPrice: Integer) = {
  ^();
  clientName = sponsor.name.copy()
};

goHome: op (homeName: Telename;
      homeAddress: Teleaddress) = {
  ....
  *.go(Ticket(homeName,homeAddress));
  *.enableEvents(PartEvent(clientName));
  here@MeetingPlace.meet(Petition(clientName));
  *.getEvent(nil, PartEvent(clientName))
};
```

リスト 2: メソッド定義例 (一部)

5.2 Agent Tcl

Dartmouth 大学で開発されたモバイルエージェントシステムである[11]．既存の Tcl 言語インタープリタを拡張して，以下のようなエージェント移動や通信に関するコマンドを導入するとともに，インタープリタ内のスタック領域やプログラムカウンタを含むプログラムの実行状態をデータ列として取り出せるようにしている．エージェント移動ではスクリプトとともに，これらの実行状態を転送する．

`agent_begin`	サーバへの登録
`agent_jump`	コンピュータ間移動
`agent_submit`	子エージェントの生成
`agent_send`	エージェント間通信
`agent_receive`	エージェント間通信
`agent_meet`	コネクション要求
`agent_accept`	コネクション受理
`agent_end`	サーバ登録を削除

以下はコンピュータ名のリスト`$machines`に含まれるコンピュータに移動して UNIX コマンドの”who”を実行し，その結果を変数`list`に格納していくものである．移動はコマンド`agent_jump`により実現する．

```
proc who machines {
  global agent
  set list ""
  foreach m $machines {
    if ([catch {agent_jump $m} result]} {
      append list
        "$m:nunable to JUMP here ($result)nn"
    } else {
      set users [exec who]
      append list
        "$agent(local-server):n$usernn"
    }
  }
  agent_send $agent(root) 0 $list
  exit
}
```

リスト 3: Agent Tcl スクリプトプログラム (一部)

Agent Tcl の特徴の一つは実行システムの構造が極めて簡単なことにある．UNIX 上の実行が前提となるが，拡張 Tcl 言語インタプリタ，エージェントの受信・送信，エージェントの管理などの主要機能が，それぞれが UNIX プロセスとして実行され，互いに UNIX のパイプ通信機構を通じて情報をやりとりしている．なお，Tcl 言語インタプリターは複数スクリプトの並行実行はできないことから，エージェントが到着したときは，送受信機構が拡張 Tcl 言語インタプリタを実行する UNIX

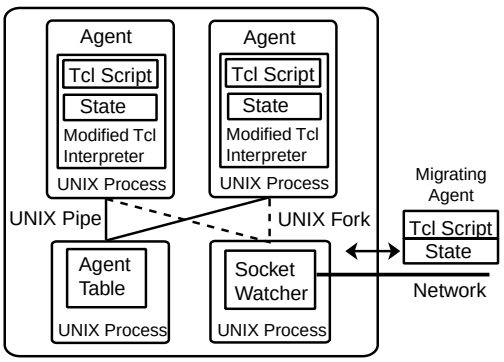

図 3: Agent Tcl のシステム構成

プロセスを新たに生成する．エージェント間通信も拡張 Tcl 言語インタプリタを実行するプロセス間のパイプ通信により実現する．また，エージェントの転送に用いるデータ通信プロトコルについても，電子メールの送信プロトコルである SMTP などを利用している．

なお，Dartmouth 大学では Agent Tcl の発展として，Java 言語などの Tcl 言語以外の記述言語にも対応可能なシステムを開発している．

5.3 Aglets

Aglets[2]は IBM の東京基礎研究所により開発されているモバイルエージェントシステムであり，エージェントだけでなくシステム自身も Java 言語により開発されている．この結果，高い移植性をもつと同時に，Java のプログラム資産が活用できるように考慮されている．

Aglets のエージェントは，主プログラムと実行状態に相当するインスタンス変数の他に，コールバックメソッド群，ユーザ定義メソッドから構成される．このうち，コールバックメソッドは Java AWT のイベントリスナーモデルに基づいて定義され，システム側から呼び出される．そのタイミングには，例えばエージェントの生成直後，移動直前，移動直後，消滅直前などがある．

表 1 : エージェント状態変化とコールバックメソッド

	状態変化直前	状態変化直後
生成		`onCreation()`
移動	`onDispatching(URL)`	`onArrval()`
消滅	`onDisposing()`	
複製	`onCloning()`	`onClone()`

ここで Aglets エージェントのプログラム例を示す．

```
public class SimpleAglet extends Aglet
implements MobileListener {
  public String name;
  // onCreation() は生成直後に呼び出される
  public void onCreation(Object init) {
    addMobilityListener(this);
    name = new String("Agent");
    try {
      // dispatch() により自分自身を移動
      dispatch("atp://some.where.com");
    } catch (Exception e) {/* 移動失敗 */}
  }
  // onDispatching() は移動直前に呼び出される
  public void onDispatch-
ing(MobiltyEvent e) {
    System.out.println(name+" is going to "
      +e.getLocation());
  }
  // onArrival() は到着直後に呼び出される
  public void onArrival(MobileEvent e) {
    System.out.println(name+" came from "
     +e.getLocation());
  }
  // 主プログラム
  public void run() {
    ....
  }
}
```

リスト 4: Aglets エージェントのプログラム例

エージェントが生成されるとメソッド `onCreation()` が呼び出され，次に主プログラムを表す `run()` が実行される．なお，インターフェース `MobileListener` は，このエージェントがメソッド `onDispatching()` と `onArrival()` を実装することを宣言し，`addMobilityListener(this)` によりリスナー登録を行うと，両メソッドがそれぞれエージェントの移動直前と移動直後に呼び出されることになる．Aglets では Telescript の `go()` に相当するコマンドとして `dispatch(URL url)` をもち，移動先は URL で指定する．上記ではコンピュータ `some.where.com` に移動することを表す．移動を行う前には `onDispatching()` が呼び出される．そして，移動先に到着するとメソッド `onArrival()` が呼び出され，次に `run()` が実行される (図 4 を参照).

エージェント移動ではエージェントを Java 言語の直列化機構を利用してデータ化している．このため，プログラムカウンターやメソッド内のローカルな変数は保存されないが，上記の `name` のようにインスタンス変数はそのまま移動先に転送され，また，処理の再開はシステムによるコールバックメソッドの呼び出しで実現する．

また，Aglets はエージェント間通信として，並行オブジェクト指向言語でみられた非同期メッセージ，同期的メソッド呼び出し，future 通信（非同期返答）を提供

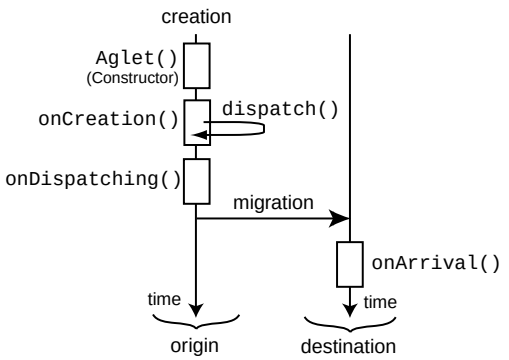

図 4: リスト 4 の Aglets エージェントの実行

している．エージェント間通信の実現では，各エージェントのプロキシーオブジェクトを介して行われ，エージェントの保護を実現している．

5.4 Voyager

ObjectSpace 社により開発された商用の分散オブジェクトシステムであり[12]，Java 言語に特化した ORB (Object RequestBroker) を中心に構成されている．この ORB を通じて，遠隔コンピュータ上への Java 言語のオブジェクト生成することや，そうした遠隔オブジェクトへのメソッド呼び出しを提供する．なお，遠隔メソッド呼び出しでは引数となるオブジェクトを通信可能な形式に直列化されて転送し，返り値も同様にオブジェクトを返すことができる．また，メソッド呼び出しの形式も，遠隔手続き呼び出しと同様な同期的メソッド呼び出しだけでなく，一方向の非同期メッセージやフューチャ通信，マルチキャスト通信なども提供する．

さらに，Voyager は従来の ORB の機能に加えて，オブジェクトのコンピュータ間移動も提供し，その特別な場合としてモバイルエージェントを実現している．また，移動したオブジェクトへのメッセージ転送も可能である．

オブジェクト／エージェントの移動は移動コマンド `moveTo()` を移動先コンピュータ (`some.where.com`) を引数にとって呼び出すことにより実現する．なお，Aglets などと同様に実行状態として，インスタンス変数を移動対象とするため，コマンド `moveTo()` 以降のプログラムは実行されない．ただし，コマンド `moveTo()` の引数として到着直後に呼び出すメソッド (`method()`) を明示的に指定することができる．現在，Voyager はモバイルエージェントを実現するシステムとして最も広く普及しているとわれる．

```
public class Traveler extends Agent {
  Sting name = null;
  // start() は外部から呼び出される
  public void start() {
    System.out.println(name+" is going to "
      +Voyager.getAddress());
    // 自分自身を移動させ，method() を呼び出す
    moveTo("some.where.com", "method");
  }
  // method は到着直後に呼び出される
  public void method() {
    System.out.println(name+" am in "
      +Voyager.getAddress());
  }
}
```

リスト 5: Voyager エージェントのプログラム例

これは，Voyager がモバイルエージェントと分散オブジェクトがもつ有用性を融合させたシステムであることに起因する．さらに，特定のインターフェースを実装していない通常のクラスでも，遠隔オブジェクト生成やメッセージ送信の対象にするための便宜が図られており，分散オブジェクトシステムとしても有用性が高い．また，Java RMI，CORBA/IIOP，DCOM などの既存の分散オブジェクト指向との親和性も考慮されている．

5.5 Plangent

Plangent[13] は東芝 S&S 研究所により開発されているモバイルエージェントシステムである．その特徴は，エージェントのコンピュータ間移動機能に加えてプランニング機能を導入している．つまり，エージェントはユーザ要求を満足する移動経路や動作内容に関するプランを生成し，そのプランに従って移動・処理を行う．ただし，処理の途中結果が十分でない場合や，ネットワークが不調な場合は，再プランニング機能を利用して，移動先コンピュータが提供する知識ベースに従ってプランを作り直すこともできる．

Plangent のエージェントは，独自の記述言語により記述されるアクション定義とスクリプト定義から構成される．

```
action(アクション名, _, _,
       [スクリプト呼び出し],
       [事前条件],
       [事後条件],
       []).
```

リスト 6: アクション定義

- **アクション定義**はプランニングのための知識を定義するものであり，(1) アクションを実行するための事前条件，(2) アクションの動作に相当するスクリプト呼び出し，(3) アクションを実行した結果を示す事後条件からなる．ここで (3) をゴールと呼び，ユーザから Prolog の項の形式でエージェントに渡される．

- **スクリプト定義**は PlangentScript と呼ぶ独自のスクリプティング言語により記述する．この言語は手続き型言語の形態をとり，制御構文として，条件文や繰り返しを表す `if` や `foreach`，失敗発生時の例外処理として `try-catch` などをもつ．また，ビルトインコマンドとして，エージェントのコンピュータ移動を表す `goto()` や，再プランニングを行う `newgoal()` などをもつ．ここで，`goto('node')` はノード名 (`node`) で表されたコンピュータへの移動を表し，`newgoal('goal')` はエージェントの新しいゴールを `goal` にして，再プランニングすることを表す．

なお，スクリプト定義中から Java 言語により記述されたプログラムを呼ぶ出す機能が提供されている．

Plangent のエージェントは新たなゴールが与えられると，自らのアクション定義を照らし合わせ，そのゴールを実現するプランを生成する．そして，そのプランに従ってスクリプト呼び出し，各種処理やコンピュータ間移動を行う．プラン実行の成功，つまり，ゴールを満足する場合は終了し，逆に失敗した場合は再プランニングを行い他のプランの実行を試みる．また，新しいゴールを再設定することもできる．

次に Plangent エージェントのプログラム例を示す．これは，複数のコンピュータに移動して，メッセージを出力するエージェントであり，それのアクション定義は以下のようになる．ここでは，ゴールとして `putHelloWorld` をもち，無条件 (大括弧のみで表記) に移動先コンピュータのリストを引数 `[gamma,beta,alpha]` として `print_hello_world` というスクリプトを呼び出す．

```
action(_, _, _,
  [print_hello_world(['gamma',
                      'beta','alpha'])],
  [],
  [putHelloWorld(['gamma','beta','alpha'])],
  []).
```

リスト 7: アクション定義例

```
print_hello_world(@nodes) {
  try {
    foreach $node (@nodes) {
      goto($node);
      print('Hello World');
    }
  } catch {
    goto('home');
    print('Failed to move');
  }
}
```

リスト 8: アクション定義例

リスト 8 は print_hello_world スクリプトを定義する．ここでは移動先のノード名のリスト (['gamma','beta','alpha']) を変数@nodes で受け取り，foreach を用いて，各ノード名を$node に取り出す．それぞれのコンピュータに移動し，拡張コマンドとして用意された print() により画面上に Hello World を表示する．ただし，try-catch を通じてスクリプト実行の失敗の判定し，失敗した場合は catch 文に制御を移し，home というコンピュータに移動し，print('Failed to move') を実行する．

これらのアクション定義やスクリプト定義はインタープリタ形式で実行される．Plangent システムは Java 言語で実装されており，エージェントの移動は Java の RMI (遠隔メソッド呼び出し) を利用して実現する．ただし，その移動において，実行中のスクリプトのスタック情報やプログラムカウンタなども情報も一緒に転送される．このため，移動先では移動直前の状態からそのままスクリプトの継続実行が可能である．

5.6 MobileSpaces

MobileSpaces[14–17]は，Aglets などと同様に Java 言語をベースにしたモバイルエージェントシステムであるが，その特徴は階層化機構を導入し，モバイルエージェントのなかに他のモバイルエージェントを入れ子状に内包できる．これにより，複数のモバイルエージェントを組織化・グループ化を通じて，粒度の大きいアプリケーションを作成可能となっている．なお，形式的な定義は MobileSpaces の計算理論を扱った論文[18]を参照されたい．

また，ランタイムシステム自体がこの階層化機構を通じて構成されており，次の 3 つの特徴をもつ．

- ランタイムシステムはエージェントの実行制御，永続化，階層構造の管理などの最小機能だけを提供し，他の機能はエージェント階層を通じてモバイルエージェントにより実現されている．従って，ランタイムシステムは OS や通信ネットワークなどの計算環境に非依存となる．
- 一方，計算環境に依存した機能や付加的な機能に対しては，それを実現するモバイルエージェントが個々に用意される．従って，エージェントのコンピュータ間移動を実現エージェントは各通信プロトコル毎に用意される．そして，これらの各種操作を実現するモバイルエージェントを移動・置換することにより，ランタイムシステム自体の動的変更や計算環境への動的に適応することができる．
- 既存のモバイルエージェントシステムは高機能化し，プログラミングが複雑化しているが，MobileSpaces ではエージェント移動を通じて各種機能を表現する．これはモバイルエージェントが必要としている機能を実現するエージェントに移動することにより実現するものである．例えば図 6 では，あるエージェントが他のコンピュータに移動が必要なときは，エージェントのコンピュータ間移動を実現するエージェントに移動して，そのコンピュータへの移動を要求する．

エージェント移動ではその処理内容や通信環境などに応じた移動経路を設定する必要があるが，経路制御自体もモバイルエージェントとして実現できる．これは他のエージェントを乗せて移動する方法と，各ホスト上に待機して到着したエージェントを別のホストに転送する方法などがある．Java 言語の直列化機構を利用しているが，電子メールを含む多様なデータ通信プロトコルに動的に対応し，さらに無線通信などの通信切断が頻発するネットワークも考慮している[19,20]．

6 研究動向

モバイルエージェントの実装に関する研究から，個々の実装・運用技術や応用事例に関心が移りつつある．ここでは最近の研究動向を紹介する．

CORBA 3.0 のオブジェクト渡し

OMG による分散オブジェクトの標準化である CORBA[21] の次期バージョン (CORBA 3.0) が標準化作業中である．この CORBA 3.0 では遠隔メソッド呼び出しの引数として，オブジェクトのコピー渡しも導入される予定である．従って，エージェントを遠隔メソッド呼び出しの引数とすることにより，その実行状態も含めた転送が容易に実現できることになる．今後

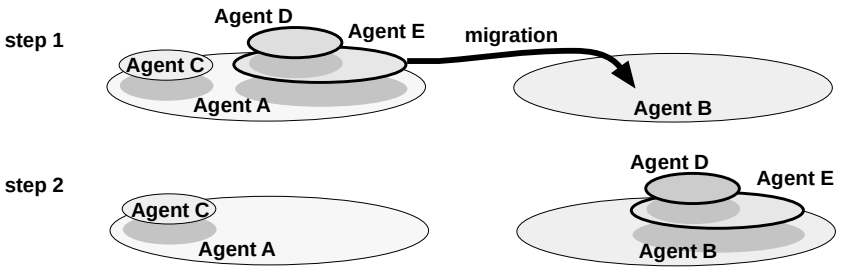

図 5: Agent Hierarchy and Inter-agent Migration

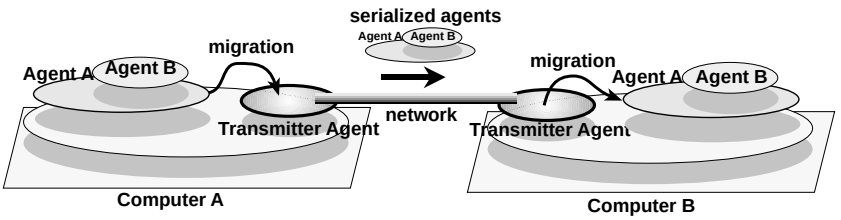

図 6: Transmitter Mobile Agents

は，CORBA 3.0 によるモバイルエージェントの実装事例が数多く登場すると予想される．

セキュリティ

不正なエージェントによるホストへの攻撃の防御だけでなく，不正なホストによるエージェントへの攻撃の防御が関心を集めている．モバイルエージェントによる電子商取引を考えた場合，エージェントが保持する情報や貨幣を不正なホストから改ざんされたり，盗まれたりする恐れがある．このほか，エージェント及びランタイムシステムの認証機構などについても盛んに研究が行われている．セキュリティの最新研究は論文集[22]を参照されたい．

モバイルエージェント の捕捉

他のコンピュータに移動したエージェントにメッセージを送信したり，連れ戻さなければいけないことがある．このため，エージェントの現在位置を捕捉する機構が提案されている．代表的な方法は，(1) 移動するごとに，移動前の場所に次の移動先アドレスを残し，それをトレースする方法，(2) 移動するごとに特定のホストに現在位置をメッセージで通知する方法，(3) 特定のホストから定期的に位置問い合わせメッセージを複数ホストに送り，返答を待つ方法などである．

デザインパターン

オブジェクト指向設計ではオブジェクトやクラスを組み合わせて開発するが，それらの組み合わせ方は数十種類のパターンに分類できるといわれている．モバイルエージェントに関しても，各エージェントプログラムのデザインパターンや，エージェントの移動経路のパターンなどが提案されている[2]．

プログラム変換による状態保持

Java 言語によるモバイルエージェントでは，Java 言語の仮想機械を変更しない限り，メソッドのローカル変数やプログラムカウンタをデータ化することは難しい．そこで，エージェントプログラムの変換を行い，メソッド内のローカル変数やプログラムカウンタに関する情報をインスタンス変数として保持させる方法がいくつか提案されている (例えば論文[23])．

モバイルエージェント 間協調

モバイルエージェントにマルチエージェントの協調手法を導入する試みがある．ただし，モバイルエージェントはその移動性から，協調相手が移動している可能性があり，それを考慮した協調手法が必要となる．

知的通信ネットワーク

通信システムとモバイルエージェントの融合も活発化している[24]．例えば，インテリジェントネットワークやアクティブネットワークなどでは，ルータやスイッチなどのネットワークノードの挙動を変更するため，通信制御するプログラムを動的にダウンロードする方法[25]や，通信パケット自体にプログラムを埋め込み，そのプログラムにより自律的に経路制御するパケット[26]などが提案されている．

7 標準化動向

モバイルエージェントの標準化には，主に二つの活動がある．

(1) FIPA

FIPA (Foundation for Intelligent Physical Agents)[27]は，1996 年に発足したエージェント技術全般に関する標準化組織であるが，1998 年にはモバイルエージェントに関しても標準化項目に加えた．エージェント移動に用いるプロトコルや，システム間で交換される情報のオントロジーなどが提案・検討されている．

(2) MASIF

MASIF (Mobile Agent System Interoperability Facilities)[28] は，CORBA[21]などの標準化作業を行っている OMG (Object Managment Group) により進められているモバイルエージェントシステム間の相互運用に関する標準化案である．具体的には，モバイルエージェントを実現するランタイムシステムに対して，エージェントの生成，移動，終了などの基本操作に関する外部インターフェース (IDL として規定）と，エージェント及びランタイムシステムを発見・登録のためののネーミングサービスに関する外部インターフェースを規定している．

補足

モバイルエージェントの標準化は時期尚早という立場をとる研究者・企業も多く，両標準化が普及するかは不確定である．さらに，FIPA はデジタルテレビ放送の標準化組織を主体として発足したため，モバイルエージェントの開発当事者が少ない．また，オントロジーなどの研究段階の技術に基礎をおくため実現性が疑問視されている．一方，MASIF はランタイムシステムの相互運用に主眼をおき，モバイルエージェント自体の標準化が不十分である．このため，エージェントが他の種類のランタイムシステムに移動しても実行できるとは限らない．

8 まとめ

モバイルエージェントは今後のネットワーク社会における必須のインフラストラクチャとなると予想されている．しかし，モバイルエージェントの研究は始まったばかりであり，数多くの課題が残されている．特に，モバイルエージェントと知識情報処理の関わりについては手付かずの状態であり，今後の研究発展が望まれている．

参考文献・学会

最後に，参考文献，学会などを紹介する．モバイルエージェントの入門としては論文[1, 29–32] がある．また，モバイルエージェントの実装技術については論文[33–35]が参考になる．モバイルエージェント／オブジェクトを主要なテーマとしている学会には，Workshop on Mobile Agents (1996 年〜)[36, 37]や，Workshop on Mobile Object Systems[38]などがある．この他，人工知能やオブジェクト指向，通信，OS 関連の学会にも論文が発表されること多く，例えば通信への応用事例については[24]が詳しい．モバイルエージェントのセキュリティ技術に関しては論文集[22] がある．

文献

[1] White, J. E., “ Telescript Technology: Mobile Agents”, *Software Agents*, Bradshaw, J. (ed.), MIT Press, 1997.

[2] Lange, B. D.; Oshima, M., “ Programming and Deploying Java Mobile Agents with Aglets”, Addison-Wesley, 1998.

[3] Strasser, M.; Baumann, J.; Hole, F., “ Mole: A Java Based Mobile Agent System”, Vitek, J. ed., *Mobile Object Systems, Lecture Notes in Computer Science*, Vol. 1222, 1997.

[4] Ousterhout, J. K,; Levy, J. Y.; Welch, B. B., “ The Safe-Tcl Security Model”, Vigana, G.(ed), *Mobile Agents and Security, Lecture Notes in Computer Science*, Vol. 1419, 1998.

[5] Necular, G.; Lee, P., “ Safe, Untrusted Agents Using Proof-Carryingy”, Rothermel, K.(ed), *Mobile*

Agents, Lecture Notes in Computer Science, Vol. 1477, 1998.

[6] Wahbe, R.; Lucco, S.; Anderson, T. E.; Graham, S. L., " Efficient Software Based Fault Isolation", *Proceedings of ACM Symposium on Operating System Principles*, pp.203-216, 1993.

[7] Tanizawa, Y.;Satoh, I.; Anzai, Y., " A Mobile Agent-Based Framework to Support Personal Mobility", to appear in *Proceedings of International Conference on Software Engineering, Artificial Intelligence, Networking & Parallel/ Distributed Computing*, August, 2001.

[8] Satoh, I., " Flying Emulator: Rapid Building and Testing of Networked Applications for Mobile Computers", to appear in *Proceedings of 5th International Conference on Mobile Agents (MA'2001), Lecture Notes in Computer Science (LNCS)*, Vol. 2240, December, 2001.

[9] Satoh, I.," MobiDoc: A Framework for Building Mobile Compound Documents from Hierarchical Mobile Agents", *Proceedings of Symposium on Agent Systems and Applications / Symposium on Mobile Agents (ASA/MA'2000), Lecture Notes in Computer Science*, Vol.1882, pp.113-125, 2000.

[10] General Magic Inc., " Odyssey Web Page", 1998, (http://www.genmagic.com/agents/odyssey.html).

[11] Gray, R. S.," Agent Tcl: A Transportable Agent System", *Proceedings of CIKM Workshop on Intelligent Information Agents*, 1995.

[12] ObjectSpace Inc., " ObjectSpace Voyager Technical Overview", ObjectSpace, Inc., 1997, (http://www.objectspace.com/Voyager).

[13] 本位田; 飯島正; 大須賀,「エージェント技術」, 共立出版, 1999.

[14] 佐藤一郎,「高階モーバイルエージェントシステム」, 情報処理学会プログラミング研究会報告, 1998.3.

[15] 佐藤一郎,「モーバイルエージェントの階層的な構成と移動」, 日本ソフトウェア科学会全国大会, pp.249-353, 1998.

[16] Satoh, I.," MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System", *Proceedings of International Conference on Distributed Computing Systems (ICDCS'2000)*, IEEE Computer Society, April, 2000, pp.161-168.

[17] Satoh, I., " An Architecture for Next Generation Mobile Agent Infrastructure", *Proceedings of International Symposium on Multi-Agent and Mobile Agents in Virtual Organizations and E-Commerce (MAMA'2000)*, ICSC Academic Press, 2000, pp.281-287.

[18] Satoh, I., " A Formalism for Hierarchical Mobile Agents", *Proceedings of Symposium on Software Engineering for Parallel and Distributed Systems (PDSE'2000)*, IEEE Computer Society, June, 2000, pp.165-172.

[19] Satoh, I., " Adaptive Protocols for Agent Migration", *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS'2001)*, IEEE Computer Society, 2001, pp.711-714.

[20] Satoh, I., " Network Processing of Mobile Agents, by Mobile Agents, for Mobile Agents", *Proceedings of 3rd International Workshop on Mobile Agents For Telecommunication Applications (MATA'2001), Lecture Notes in Computer Science*, Vol. 2164, pp.81-92, August, 2001.

[21] Object Management Group, " The Common Object Request Broker: Architecture and Specification, Revision 2.0", *OMG formal document*, 97-02-25, 1997.

[22] Vigana, G.(ed)," Mobile Agents and Security", *Lecture Notes in Computer Science*, Vol. 1419, 1998.

[23] Funfrocken, S., " Transparnt Migration of Java-Based Mobile Agents", Rothermel, K.(ed), *Mobile Agents, Lecture Notes in Computer Science*, Vol. 1477, pp.26-37, 1998.

[24] Karmouch, A.(ed), " Mobile Software Agents for Telecommmunications", *IEEE Communications Magazine*, Vol. 36, No.7, pp.24-85, 1998.

[25] Wetherall, D. J.; Guttag, J. V.; Tennenhouse, D. L., " ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols", *Proceedings of International Conference on Open Architectures and Network Programming*, April 1998.

[26] Gunter, C. A.; Nettles, S. M.; Smith, J. M., " The SwitchWare Active Network Architecture", *IEEE Network, special issue on Active and Programmable Networks*, Vol. 12, No. 3, 1998.

[27] Foundation for Intelligent Physical Agent, " Agent Management Support for Mobility", *FIPA98 Draft Specification*, 1998, (http://drogo.cselt.it/fipa).

[28] Object Management Group,“ Mobile Agent Facility Specification”, *OMG TC Document*, ORBOS/97-10-05, 1997.

[29] Chess, D. M.; Harrison, C. G.; Kershenbaum, A., “ Mobile Agents: Are They a Good Idea?”, Rothermel, K.(ed), *Mobile Agents, Lecture Notes in Computer Science*, Vol. 1219, pp.25-45, 1997.

[30] 佐藤一郎,「モバイルエージェントの動向」, 人工知能学会論文誌, Vol.14, No. 4, pp.598-605, 1999.

[31] 佐藤一郎,「FAQ: モバイルエージェント」, コンピュータソフトウェア, Vol.17, No.1, pp.24-25, 2000.

[32] 佐藤一郎,「モバイルエージェント」, コンピュータソフトウェア, Vol.17, No.2, pp.45-54, 2000.

[33] Milojicic, D. S.; Douglis, F.; Paindaveine, Y.; Zhou, S., “ Process Migration”, *ACM Computing Surveys*, Vol. 32, No. 3, pp.241-299, 2000.

[34] Nelson, J., “ Programming Mobile Objects with Java”, Wiley, 1999.

[35] Thorn, T., “ Programming Languages for Moible Code”, *ACM Computing Surveys*, 29(3), pp.213-239, 1997.

[36] Rothermel, K.(ed), “ Mobile Agents”, *Lecture Notes in Computer Science*, Vol. 1219, 1997.

[37] Rothermel, K.(ed), “ Mobile Agents”, *Lecture Notes in Computer Science*, Vol. 1477, 1998.

[38] Vitek, J., “ Mobile Object Systems”, *Lecture Notes in Computer Science*, Vol. 1222, 1997.