チュートリアル

モバイルエージェント

佐藤 一郎

1 はじめに

次世代の分散処理技術としてモバイルエージェントが注目され，これを実現するソフトウェアが数多く提案・公開されている．本稿では，モバイルエージェントについて概説するとともに，これらのソフトウェアのいくつかを紹介していく．

モバイルエージェントは，ネットワーク上のコンピュータを移動しながら処理を進めるプログラムである．その特徴として，プログラムコードだけではなく，プログラムの実行状態，つまり変数内容なども同時に移動し，その移動先では移動前の実行状態から処理を再開することができる．

ネットワークを移動するプログラム技術はモバイルエージェントがはじめてではない．例えば1980年代中頃には，プロセスマイグレーションなどの名称で，プログラムの移動技術が盛んに研究された．しかし，これらの既存技術は，負荷分散や耐故障性の実現に主眼をおき，実行中プログラムを計算負荷の重いコンピュータから軽いコンピュータに受動的に移動させることが主な目的となっていた．一方，モバイルエージェントはその移動性を積極的に利用して，高次な分散処理を実現することに主眼をおいている．さらに，モバイルエージェントは自律的な計算実体として導入され，エージェント自身が移動先や移動タイミングを決定しながら，主体的に他のコンピュータに移動することが前提となる．

こうしたモバイルエージェントの概念の多くは，1990年代中頃に登場するGeneral Magic社のTelescript [14] の影響を強く受けている．Telescriptはモバイルエージェントによる電子商取引を目標とした商用システムであり，エージェントプログラム中でgo()コマンドを呼び出すことにより，そのエージェント自身を他のコンピュータに移動することができた．当時は話題先行の部分も多かったが，Telescriptはその後のモバイルエージェント研究の先駆けとなった．そして，1997年頃になるとJava言語 [1] に後述する直列化機構(serialization)が導入され，Javaオブジェクトのネットワーク転送が容易になり，Java言語によるモバイルエージェントが数多く提案されるようになった．

モバイルエージェントはエージェントという名称がつくが，他のエージェント技術とは異なり，人工知能的な特性は要求されないことが多い．また，モバイルエージェントはオブジェクト指向プログラミング言語により記述することが多く，モバイルオブジェクトの一種として扱われることがある．しかし，モバイルエージェントは能動性や主体性などを中心に，従来のモバイルオブジェクトの枠組みを越える概念を含んでいる．また，プログラム粒度の観点からも，モバイルオブジェクトはプログラム部品なども対象とするが，モバイルエージェントでは自己完備性が要求されることが多い．

本稿では2章でモバイルエージェントを概説し，3章ではモバイルエージェントを実現するソフトウェアとし

Mobile Agents.

Ichiro Satoh, お茶の水女子大学理学部情報科学科 / 科学技術振興事業団 さきがけ研究21, Ochanomizu University/PRESTO, Japan Science and Technology Corporation (JST).

コンピュータソフトウェア, Vol.17, No.2 (2000), pp.45–54.
[チュートリアル] 1999年8月5日受付.

て Telescript [14], Agent Tcl [2], Aglets [7], Voyager [9], Plangent [5], MobileSpaces [10]などを紹介する．4章ではまとめとともに，モバイルエージェントの今後の動向について概観する．なお，モバイルエージェントの特性や応用事例，標準化については別稿 [11] を参照されたい．

2 モバイルエージェントシステムの機能と構成

ここではモバイルエージェントを実現するソフトウェアの構成や機能について概説する．モバイルエージェントの移動や実行を行うには，ランタイムシステムやエージェントプラットフォームなどと呼ばれるソフトウェアをコンピュータ上に予め動作させておく必要がある (図1)．これはエージェントの実行を管理・制御するとともに，エージェントの送受信を行うソフトウェアである．また，エージェントプログラムの実行は，インタープリタまたは仮想機械方式によることが多く，それらのソフトウェアも必要となる．

モバイルエージェント記述言語

既存のモバイルエージェントシステムは，エージェント記述に独自のプログラミング言語を導入するものと，Java や Tcl, Scheme, ML などの既存プログラミング言語を利用するものに大別される．前者はモバイルエージェントに最適な言語となるように設計・実装することができるが，後者は便宜的な言語拡張やライブラリを通じて，モバイルエージェント特有の機能を導入することから，記述性に制約が多い．しかし，言語習得の容易さやソフトウェア資産の再利用の観点から後者を用いるものが多く，さらに近年は Java 言語を利用したものが多くなっている．

モバイルエージェントの解釈・実行

移動先コンピュータのハードウェアや OS は相違する可能性がある．また，ネイティブコード形式でモバイルエージェントを直接実行することは，不正アクセスなどのセキュリティ上の問題が発生しやすい．このため，エージェントのプログラムをハードウェアや OS に依存しない中立な形式で記述し，仮想機械またはインタプリタにより解釈・実行する方法が広く利用されている．なお，ランタイムシステム自体も仮想機械上で稼働させて，移植性を高めることが多い．

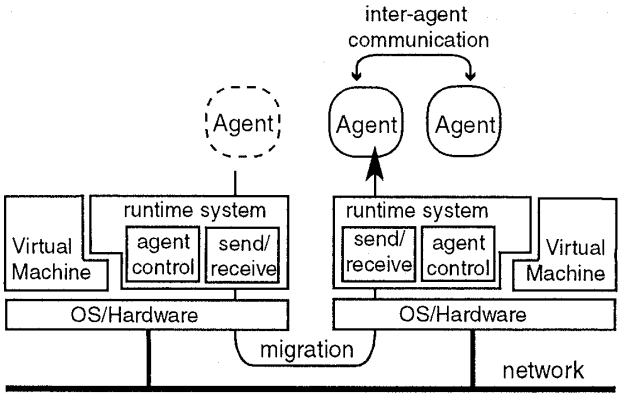

図1 モバイルエージェントシステムの構成

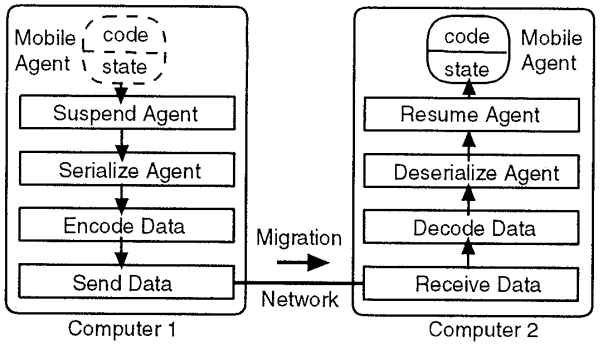

図2 エージェント移動

モバイルエージェントの移動

Java Applet や Postscript などはプログラムをクライアントコンピュータやプリンタに転送して実行する．ただし，これらはプログラムコードだけを転送するのに対し，モバイルエージェントでは実行中プログラムの実行状態も転送対象となる．この実行状態の転送は RPC(遠隔手続き呼び出し) と同様の仕組みを用いることが多く，図2のように移動前に変数などのメモリ上の情報を，ネットワークに転送可能なデータ形式に変換する[†1]．そして，このデータ列を一般のデータ通信としてネットワーク上に転送する．移動先では到着したデータ列を再びメモリ上のイメージに変換して，エージェントとして実行を再開する．

補足: 移動方式

モバイルエージェントの移動に際して，転送される実行状態の範囲はシステムによって異なるので注意が必要である．既存の方式は以下の2つに大別される．

(A) ヒープ領域内の情報だけを転送対象とする方式

†1 直列化 (serialization) または整合化 (marshalling) と呼ぶことが多い．

(Weak Migration)

(B) 方式 (A) の範囲に加えて，スタック領域内の情報やプログラムカウンタも転送対象とする方式 (Strong Migration)

方式 (A) はオブジェクト指向プログラミング言語におけるインスタンス変数だけを転送対象とするものに相当し，メソッド内のローカル変数や実行箇所などの情報は対象外となる．従って，ループや再帰などを実行している最中に移動しても，そのまま移動直前の実行箇所から処理を再開することは難しく，移動先では所定のメソッドや関数を明示的に呼び出す必要がある．一方，方式 (B) では任意のタイミングで移動させられても，移動直前の実行箇所からそのまま処理を継続することができる[†2]．しかし，スタック領域内の情報は膨大になることが多く，それを保存・転送するコストは大きい．

これまでのプロセスマイグレーションやモバイルオブジェクトは，負荷分散などを目的にプログラムを移動させることが多く，移動等価性，つまり任意の実行箇所で移動しても処理の再開ができる方式 (B) を用いる必要があった．しかし，モバイルエージェントはそれ自身により主体的に移動することが多く，移動タイミングもエージェント自身で決定できる．このため，任意の実行箇所で処理を再開する必然性は少ない．また，実際的なアプリケーションでは方式 (A) で十分であるという研究報告が多い (例えば [12]を参照)．特に，ファイルやウィンドウなどの何らかの計算リソースを利用するエージェントでは，移動前後にそれらを明示的に解放・獲得する必要があり，方式 (B) であっても移動前後に例外的処理を行う必要が生じる．このため，方式 (B) のメリットを活かせないことが多い．

なお，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おり，(1) 予め移動先にプログラムコードを配布しておく方式，(2) エージェント実行中に必要に応じてコードをオンデマンドに転送する方式，(3) 実行状態とともに必要なコードを一括転送する方式などがある．ただし，各方式の有効性は転送効率や通信・計算環境などに依存する．

†2 ただし，複数の実行スレッドをもつプログラムは対象外となることが多い．

3 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．

3.1 Telescript

Telescript [14] (図 3) 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で移動させ，遠隔データベースの情報収集や電子商取引などの実現を目的に開発された．Telescript は次の概念により特徴づけられる．

- **プレース** (Place) 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** (Travel) 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，移動先のプレースは相違なコンピュータ上のものでもよい．エージェントが移動するときはコマンド `go()` を実行する．ただし，その引数としてチケットと呼ぶ特別なオブジェクトが必要となる．チケットは，移動先プレース名に加えて，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からなり，移動に関する諸条件を定義するものである．
- **ミーティング** (Meeting) は同じプレースに存在するエージェント同士の通信である．エージェントは

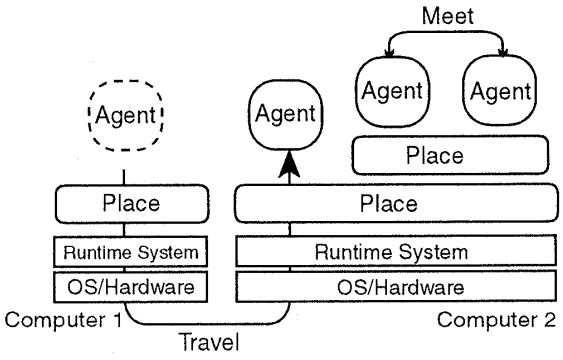

図 3 Telescript エージェント

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．引数には通信相手名や通信開始時間などを指定する．要求が受け入れられると，通信が開始され，互いのメソッドを呼び出すことができる．そして，通信当事者の一方がコマンド part() を実行することにより通信が終了する．

- **オーソリティ** (Authority) 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に加えて，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から構成される．これは Telescript の応用事例の１つが電子商取引であり，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ためである．
- **パーミット** (Permit) はエージェントやプレースに与えられ，それらの能力を表す．例えば，特定命令の実行許可，エージェントの生存時間，CPU 消費量，プログラムサイズなどの上限を定めている．そして，生存時間や CPU 消費量が上限を越えると，そのエージェントは破棄される．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もできる．なお，このパーミットを通じて，特定のエージェントが不正アクセスだけでなく，計算資源を不当に消費することも防げる．

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により記述される (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，エージェントの格納機構，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る．また，移動に際しては前述の方式 (B) を用いるため，コードだけでなく，ヒープやスタック領域内の情報，プログラムカウンタも移動し，移動先では go() コマンドの直後からそのまま処理を再開することができる．

```
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)

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 を開発した．

3.2 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 (図 4) である [2]．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，インタープリタの実行状態も転送する (方式 (B))．この結果，移動先では移動コマンドの直後から処理を再開することができる．

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

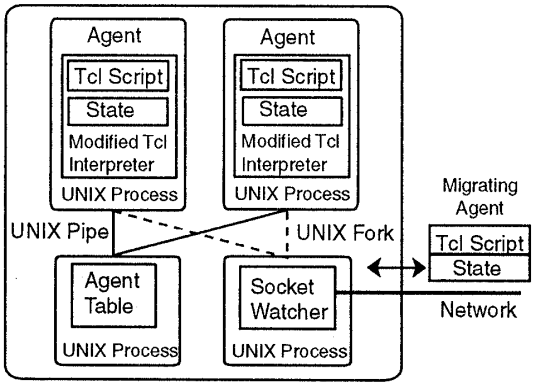

図4 Agent Tcl のシステム構成

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1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・送信機能，エージェントの管理機能が，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すると，送受信機構が拡張Tcl言語インタプリタを実行するためにUNIXプロセスを新たに生成して，複数エージェントの同時実行を可能にする．このとき，エージェント間通信も拡張Tcl言語インタプリタ間のUNIXパイプ通信により実現する．なお，エージェントの転送に用いるデータ通信プロトコルは，TCP/IPのほか電子メールの送信プロトコルであるSMTPなどが利用できる．

なお，Dartmouth大学ではAgent Tclの発展として，D'Agentsとよぶ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[3]．これは各コンピュータ上で稼働させるモバイルエージェントの実行システム上にTcl言語以外にJava言語などの複数言語の仮想機械やインタプリタを稼働させて，異なる言語で記述されたエージェントの移動と実行を可能にするシステムである．

3.3 Aglets

Aglets [7]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る．システム自体がJava言語により実装されており高い移植性をもっている．また，エージェントもJavaオブジェクトとして導入される．

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の委譲イベントモデル(Delegation-based Event Model)と同様なものであり，イベントリスナーとして明示的に登録したエージェントに対して，表1のようなタイミングでシステム側から呼び出される．そして，これらのコールバックメソッドをユーザが再定義することにより，イベントに対応した処理を指定することができる．なお，このコールバックメソッドを主体とするモバイルエージェントプログラムの構成方法はその後のモバイルエージェントプログラミングに大きな影響を与えた．

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 {
```

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()

```
  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 onDispatching(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，次に主プログラムに相当する `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する．そして `addMobilityListener(this)` によりリスナー登録を行うと，両メソッドが図5のようなタイミングで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 で表記する．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．

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，前述の方式 (A) に従うことになり，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，到着後の処理の再開はシステムにより所定のコールバックメソッドを呼び出すことで実現する．

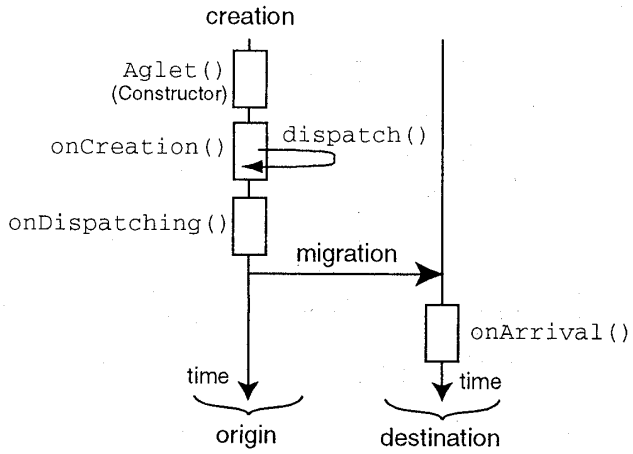

図 5 Aglets エージェント (リスト 4) の実行

このほか，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，フューチャー通信を提供している．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．

3.4 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 [9]，Java 言語に特化した ORB (Object Request Broker) 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に Java 言語のオブジェクトを生成することや，遠隔オブジェクトのメソッドを呼び出すことが可能になる．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前述の方式 (A) に従って通信可能な形式に直列化して転送し，返り値も同様にオブジェクトを返すことができる．メソッド呼び出しの方法は，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，フューチャ通信，マルチキャスト通信なども提供している．

さらに，Voyager は従来の ORB の機能に加えて前述の方式 (A) に基づくオブジェクトのコンピュータ間移動機能や，移動したオブジェクトへのメッセージ転送機能を提供している．

```
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

オブジェク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．ただし，前述の方式 (A) に従うため，コマンド `moveTo()` 以降のプログラムは実行されない．

なお，Voyager ではモバイルエージェントとモバイルオブジェクトの厳密な区別はない．ただし，モバイルオブジェクトをモバイルエージェントとして明示的に定義することができる．この場合，モバイルエージェントとなるオブジェクトは能動的に実行することができ，さらに前述のコマンド `moveTo()` の引数として，到着後に実行すべきメソッドを明示的に指定できるようになる．

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は最も広く普及しているといわれている．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り，既存の分散アプリケーションの移植が容易となるからである．この他，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できるなど，分散オブジェクトシステムとしても利便性が図られている．さらに，CORBA などの既存の分散オブジェクトとの親和性も考慮されている．

3.5 Plangent

Plangent [5] 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に加えて，プランニング機能を導入していることである．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，再プランニングを行うことができる．

Plangent のエージェントはアクション定義とスクリプト定義から構成される．

- **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の呼び出し，(3) アクションを実行した結果を示す事後条件などからなる．ここで (3) をゴールと呼び，ユーザから Prolog の項形式でエージェントに渡される．

```
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

- **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 `while`, `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行う `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拡張コマンドとして呼び出す機能も提供されている．

Plangent のエージェントの動作は次のようになる．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と照らし合わせ，そのゴールを実現するプランを生成する．次に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．そして，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．さらに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モバイルエージェントであり，それのアクション定義はリスト 7 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) で print_hello_world というスクリプトを呼び出す．なお，その引数として移動先コンピュータのリスト [gamma,beta,alpha] を与える．

```
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: スクリプト定義例

リスト 8 は print_hello_world スクリプトを定義している．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 文を用いて，各ノード名を変数 $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 文により，スクリプト実行が失敗した場合は catch ブロック内に制御を移し，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，前述の方式 (B) に従い，その移動においてはインタプリタの実行状態も転送することから，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の諸情報も一緒に転送され，移動直前の実行箇所からそのままスクリプトの実行を再開することができる．

3.6 MobileSpaces

MobileSpaces [10]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．その特徴はモバイルエージェントの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することができる (図 6)．これにより，複数のモバイルエージェントの組織化やグループ化することが可能になり，複雑かつ大規模なアプリケーションも容易に作成できるようになっている．

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，マイクロカーネルアーキテクチャを模して構成されており，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，ローダ，階層構造の管理などの最小限の機能だけを提供する．なお，ランタイムシステム自体がエージェント階層のルートとなり，他のエージェントは階層上に配置される．ただし，ランタイムシステムはそれのエージェント階層内のエージェント移動は提供するが，他のコンピュータへの移動や永続化などの計算環境に依存した機能は提供しない．
- 一方，計算環境に依存した機能や付加的な機能は，それぞれの機能を実装するモバイルエージェントを通じて実現する．そして，これらの各種機能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，システム自体の機能追加・変更を可能にする．
- モバイルエージェントに対する操作は煩雑になることが多いが，MobileSpaces ではこれらの操作をエージェント階層内の移動として統一的に取り扱うことができる．つまり，あるエージェントに対して操作が必要となったときは，その操作を実装するエージェント内に，その対象となるエージェントを移動させて実現する．

例えば，エージェントの永続化では，永続化機能を実装するモバイルエージェントが階層内に用意され，これに永続化対象のエージェントを移動させて，永続化を実現する．図 7は，MobileSpaces におけるコンピュータ

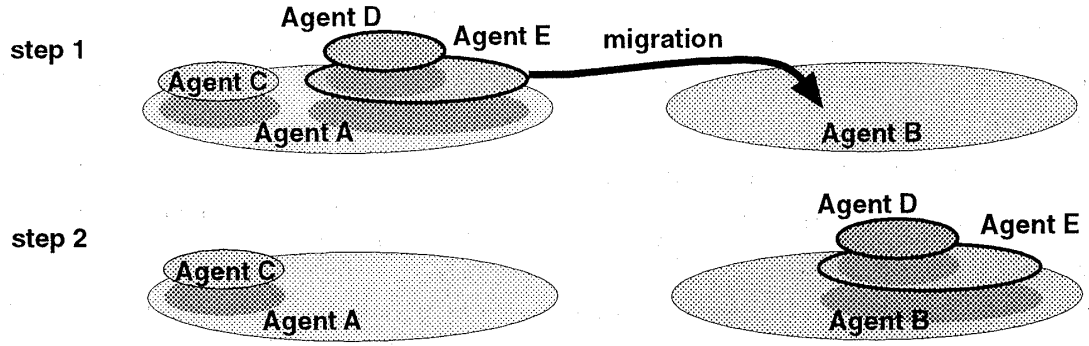

図6 MobileSpaces の階層エージェント

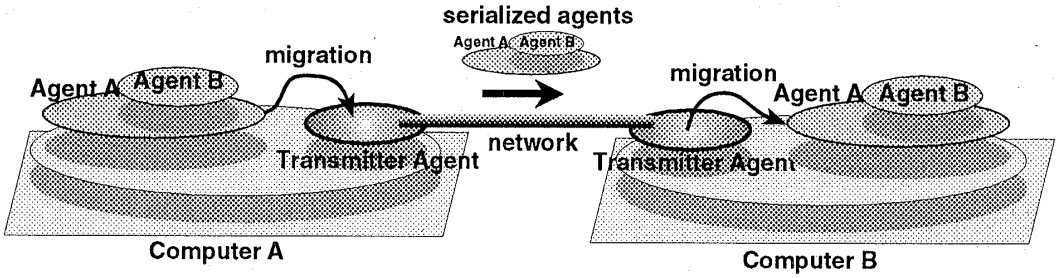

図7 MobileSpaces におけるコンピュータ間移動の実現

間移動を図示したものである．ここでは，コンピュータ間移動を実現する転送エージェントをそれぞれのコンピュータに用意している．これらの転送エージェントはその内部のエージェントをもう一方のコンピュータの転送エージェントに送るものである．そして，移動対象のエージェントを階層内移動を通じて，この転送エージェントに移動させて，コンピュータ間移動を依頼する．なお，エージェントのコンピュータ移動では Java 言語の直列化機構を利用しているが (方式 (A))，各エージェント転送プロトコルに対応した転送エージェントを複数用意し，それを選択することにより，エージェント移動プロトコルを動的に変更することもできる．

4 まとめ

上記で紹介した以外にも数多くのモバイルエージェントシステムが提案されている．また，今後も新しいシステムが数多く登場すると思われる．

その方向性の 1 つは Voyager にみられるような分散オブジェクトとの融合である．分散オブジェクトの代表的な標準化として CORBA [8] があげられるが，それの次期バージョンである CORBA version 3 では，遠隔メソッド呼び出しの引数として，オブジェクトの参照 (Object-By-Reference) だけでなく，オブジェクトのコピー (Object-By-Value) も受け渡し可能になる．これにより，遠隔メソッド呼び出しの引数や返り値としてモバイルエージェントが転送できることになり，CORBA version 3 によるモバイルエージェントシステムの実装や，逆に CORBA のアプリケーションにおいてモバイルエージェント手法が利用される可能性がある．

この他，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 [6]．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機器の挙動を変更するため，通信制御プログラムをネットワーク機器に動的に転送・実行する方法が提案されている (例えば [13])．この他，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が提案されている (例えば [4])．これらの技術はモバイルエージェントと近く，両者の融合が試みられている．この他，情報家電分野において，家電ネットワーク上の各種機器やセンサの遠隔制御・監視にモバイルエージェントを応用する試みもある．

モバイルエージェントに人工知能手法を導入した代表

例として Plangent があげられる．しかし，その他の研究事例は多いとはいえない．一方，モバイルエージェントは多様なコンピュータに移動して処理を行うが，移動先の計算環境を事前に予測することは困難である．このため，エージェント自身が計算環境へ柔軟に対応することや，移動経路をエージェント自身が動的に生成できることが必要となる．このため，モバイルエージェントに人工知能手法の導入は重要であり，今後の研究発展が望まれている．

最後に，本稿で紹介したソフトウェアの入手先を紹介する (URL 情報は 1999 年 11 月 15 日時点)．

- Agent Tcl/D'Agent (Dartmouth 大) :
  http://agent.cs.dartmouth.edu
- Aglets (アイビーエム):
  http://www.trl.ibm.co.jp/aglets
- Voyager (ObjectSpace):
  http://www.objectspace.com
- Plangent (東芝):
  http://www2.toshiba.co.jp/plangent
- AgentSpace/MobileSpaces (お茶の水女子大):
  http://islab.is.ocha.ac.jp/agent

参考文献

[1] Arnold, K. and Gosling, J. : *The Java Programming Language*, Addison-Wesley, 1996.

[2] Gray, R. S. : Agent Tcl: A Transportable Agent System, *Proc. CIKM Workshop on Intelligent Information Agents*, 1995.

[3] Gray, R. S., Kotz, D., Cybenko, G. and Rus, D. : D'Agent: Security in a Multiple Language Mobile-Agent System, *Mobile Agents and Security*, LNCS Vol. 1419, pp. 154–187, 1995.

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

[5] 本位田真一，飯島正，大須賀昭彦：エージェント技術，共立出版，1999.

[6] Karmouch, A. : Mobile Software Agents for Telecommunications, *IEEE Communication Magazine*, Vol. 36 No. 7, 1998.

[7] Lange, B. D. and Oshima, M. : *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley, 1998.

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

[9] ObjectSpace Inc : *ObjectSpace Voyager Technical Overview*, ObjectSpace, Inc. 1997.

[10] Ichiro Satoh : MobileSpaces: A Framework for Building Adaptive Distributed Applications using a Hierarchical Mobile Agent System, to appear in *Proc. International Conference on Distributed Computing Systems*, April, 2000.

[11] 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.

[12] Strasser, M., Baumann, J. and Hole, F. : Mole: A Java Based Mobile Agent System, *Proc. ECOOP Workshop on Mobile Object Systems*, pp. 301–308, June, 1997.

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

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