プログラミング 18－6
(1998. 3. 23)

AgentSpace: 高階モーバイルエージェントシステム

佐藤 一郎

お茶の水女子大学 理学部 情報科学科

東京都 文京区 大塚 2-1-1
Tel: 03-5978-5388　　Email: ichiro@is.ocha.ac.jp

あらまし

本論文では高階モーバイルエージェントシステムを提案する。これは Cardelli らによる Mobile Ambients の概念を導入したものであり、このシステムにおけるモーバイルエージェントは、コンピュータ間の移動能力をもつ能動的な計算実体であると同時に、その内部に他のモーバイルエージェントを内包することができる。これにより、エージェントの移動を通じて、複数エージェントの動的な組織化やグループ化が可能になる。また、エージェントは、それが内包するエージェントが利用する通信方式、各種サービス、制御方法をメタ定義できるという特徴をもつ。本論文では、提案する高階モーバイルエージェントの基本概念を示し、さらにそれの実現システムについて概説する。

AgentSpace: a Higher Order Mobile Agent System

Ichiro Satoh

Department of Information Sciences, Ochanomizu University

2-1-1 Otsuka Bunkyo-ku Tokyo, 112
Tel: +81-3-5978-5388　　Email: ichiro@is.ocha.ac.jp

ABSTRACT

This paper proposes a new mobile agent system which can migrate higher order agents from one machine to another. The system is inspired by the concept of Cardelli's Mobile Ambients. It is characterized in that each agent is an autonomous computational entity and can nest other agents, as well as its own data and live computation. It allows agents to be organized hierarchically and dynamically. Also, each agent can define how its inner agents communicate with one another. Therefore, an agent can change the style of interactions with other agent by moving to other agents. This paper presents some basic ideas of the system and an implementation of it.

1 はじめに

近年、モーバイルエージェントシステムなどの名称により、コンピュータ間移動能力をもつ自律的プログラムを実現するシステムが数多く提案され、次世代の分散計算システムの枠組として注目されている[1]。しかし、実際的応用を考えた場合、既存のシステムには克服すべき問題が幾つか残されている。

例えば、既存のモーバイルエージェントシステムでは、エージェントというそれぞれ独立したプログラムが移動や計算の実行単位となるが、その一方で、複数のエージェントを一つに合成する手法が欠けている。このため、オブジェクト指向計算などで多用される手法、つまり、複数の部品オブジェクトの集合体も一つのオブジェクトとして扱うという階層的手法がモーバイルエージェントでは利用できないことになる。

また、エージェント間の通信方法や、エージェントとそれの実行システム間の各種サービスや制御は、システムにより事前に定められた方法を利用する必要があり、柔軟性に欠けている。例えば、エージェントの移動や永続化の際には、システムはその前後において、エージェントに用意された特定のコールバックメソッドを呼び出し、エージェントに状態変化を伝えるように設計されていることが多い。しかし、これらのコールバックメソッドの名前やデータ型、その呼び出しタイミングはシステムにより固定化されている。また、エージェント間の通信方式は、(非)同期通信などの単純な方式しか提供されていないことが多く、エージェントの処理内容に即した協調動作を実現する上で、大きな障害となっている。

本論文では、上記の問題の解決するため、モーバイルオブジェクト／モーバイルエージェントの理論的計算モデルの一つである Mobile Ambients [4] の計算メカニズムを導入したモーバイルエージェントシステムを提案する[2]。以下、次節において Mobile Ambients を概説し、3節ではそれによる既存のシステムの問題点に対するの解決方法を議論する。4節では提案するシステムの構成を、また、5節では実装概要と評価を述べる。6節では関連研究を概観し、最後に7節でこの論文のまとめを行う。

[1]従来のプロセス／オブジェクト移動技法に対して、モーバイルエージェント [7] は移動対象となるプログラムの能動性や自律性がその特徴となる。ただし、本論文では、エージェントは単なる能動的プログラム／オブジェクトであるとし、知能などは要求しない。

[2]本論文の目的は、Mobile Ambients 自体の実装ではなく、既存のモーバイルエージェントシステムの問題点を解決する方法として、Mobile Ambients の概念を利用することにある。

2 Mobile Ambients

Mobile Ambients [4] は、L.Cardelli と A.D.Gordon によって提案された移動プログラムのための計算モデルであり、既存のプロセス代数／計算と同様な、操作意味論に基づく項書き換えシステムとして定式化されている。ただし、既存のプロセス代数／計算がプロセス間通信を基本計算メカニズムにしているのに対し、Mobile Ambients は能動的プログラムの移動を基礎としている[3]。なお、Mobile Ambients は λ 計算のコーディング能力や各種の並列システムに対する表現性がすでに明らかになっている。以下に Mobile Ambients の構文(一部)を示す。

$$P \ ::= \ n[P] \ \mid \ (\nu n)P \ \mid \ P_1 \mid P_2 \ \mid \ \mathbf{in}\, n\,.P \ \mid \ \mathbf{out}\, n\,.P$$

ここで $n[P]$ は Ambient と呼ばれる計算実体であり、プロセス代数／計算におけるプロセスに相当する。ここで、n は名前またはケーパビリティであり、そして P も Ambient であることから、Ambient の内部に別の Ambient を内包できることに注意。$(\nu n)P$ は名前またはケーパビリティ n を P 内にカプセル化することを表す。また、$P_1 \mid P_2$ は P_1 と P_2 が並列に動作することを表す。ここで $P_1 \mid P_2$ と $P_2 \mid P_1$ の意味的な違いはない。

Ambient の移動 (in)： $\mathbf{in}\, n\,.P$ は名前 n をもつ Ambient に入ることを表し、例えば次のような遷移を行う。

$$n[\mathbf{in}\, m\,.P_1 \mid P_2] \mid m[P_3] \ \rightarrow \ m[n[P_1 \mid P_2] \mid P_3]$$

ここで、$n[\mathbf{in}\, m\,.P_1 \mid P_2] \mid m[P_3]$ は二つの Ambient の $n[\mathbf{in}\, m\,.P_1 \mid P_2]$ と $m[P_3]$ が並列動作することを表す。そして、前者は名前が n の Ambient の中で二つプログラム $\mathbf{in}\, m\,.P_1$ と P_2 が並列に動作することを表している。一方、後者は名前が m の Ambient の中でプログラム P_3 が動作していることを表す。そして、上記の遷移は、前者が $\mathbf{in}\, m\,.P_1$ により名前 n をもつ Ambient が、名前が m の Ambient の中に入ることを意味する。

Ambient の移動 (out)： $\mathbf{out}\, m\,.P$ は名前 m をもつ Ambient から出ることを表す。これの遷移例として $m[n[\mathbf{out}\, m\,.P_1 \mid P_2] \mid P_3]$ を考える。これは m と

[3] π 計算 [11] における Mobility とは並行プロセス間の通信ポートの受け渡しを指し、プロセス自体の移動性は考慮されていないことに注意。

いう名前をもつ Ambient に、二つの Ambient である $n[\mathbf{out}\, m\, .P_1 \mid P_2]$ と P_3 が入っていることを表す。前者は名前 n をもつ Ambient を表し、その内部では二つのプログラム $\mathbf{out}\, m\, .P_1$ と P_2 が並列に動作することを示している。

$$m[n[\mathbf{out}\, m\, .P_1 \mid P_2] \mid P_3] \quad \rightarrow \quad n[P_1 \mid P_2] \mid m[P_3]$$

上記の遷移では $\mathbf{out}\, m\, .P_1$ をもつ Ambient である $n[P_1 \mid P_2]$ が名前 m の Ambient から出ることになる。

Mobile Ambient とモーバイルエージェントの相違:
既存のモーバイルエージェントの枠組みと比較したとき Mobile Ambient は次のような特徴をもつ。

- **階層化**： Ambient は、エージェントと同様に能動的なプログラムであると同時に、Ambient の内部に一つ以上の Ambient を入れ子状に含むことができる。
- **主体的移動**: Ambient は $\mathbf{in}\, n\, .P$ または $\mathbf{out}\, n\, .P$ を通じて主体的移動する。ただし、その Ambient 自体を内包する外側の Ambient が移動するときは、それとともに移動する。このため、主体的移動する場合を除いて、Ambient の計算環境、つまりそれの外側の Ambient が変化することはなく、計算の継続性が保証される。

これらの特徴は、複数モーバイルエージェントの階層化や合成を実現する上でも有効となる。

3 方針

この論文では、既存のモーバイルエージェントシステムの問題点を解決する方法として、Mobile Ambients の基本計算メカニズムを導入し、それに基づくモーバイルエージェントシステムを提案する。具体的には以下の概念を導入したモーバイルエージェントシステムを考える。

高階モーバイルエージェント

Ambient と同様に、モーバイルエージェントは能動的な計算実体であり、その内部に複数のモーバイルエージェントが入れ子状に内包されることがある。そして、モーバイルエージェントは自律的に他のモーバイルエージェントに移動することができるが、このとき、その内部にエージェントを内包している場合はそれらのエージェントも同時に移動し、内包関係を保存する。

これにより、モーバイルエージェントは他のモーバイルエージェントに移動することを通じて、モーバイルエージェント同士の階層化・グループ化や、モーバイルエージェント合成体のの構造変更を実現する。逆にエージェントがそれを内包するエージェントから出ることによりエージェント合成体の分解を行う。

なお、**高階**モーバイルエージェントと呼ぶのは、ここで提案するシステムの目的が、モーバイルエージェントの単なる階層化でなく、エージェントの通信や制御方法、計算環境が、そのエージェントを内包する外側エージェントによりメタ定義されることにある [24]。次に、これらの高階性を利用した特徴について述べる。なお、以降ではモーバイルエージェントを単にエージェントと呼ぶことがある。

エージェント間通信

エージェント通信はそれを内包する外部エージェントを介して行うとする。そして、その通信方法は外部エージェントに用意されたメッセージ配送プログラムによりメタ定義され、多様なエージェント間通信に対応できるようにする。また、エージェントは別の外部エージェントに移動することにより、エージェント間の通信方式を動的に変更できるようにする。

エージェントの制御・サービス

エージェントが移動・永続化される際には、その前後にエージェントに用意された所定のコールバックメソッドが呼び出される。ただし、呼び出されるコールバックメソッドの名前、データ型、タイミングはそのエージェントを内包する外部エージェントによりメタ定義可能とする。また、エージェントが利用する各種リソース情報などのサービスは、それを内包する外部エージェントにより与える。これにより、エージェントの制御・サービス方式をシステムより固定化されず、外部エージェントにより再定義可能となるようにする。

エージェントの移動・内包条件

エージェントは他のエージェントに移動可能であるが、任意の到着エージェントを内包することは、協調関係がないエージェントも取り込む危険性をもたらす。特に、内包されるエージェントの制御に利用するコールバックメソッドを、到着したエージェントが持っていないと、内包する側である外側エージェントは、そのエージェントの制御ができなくなる。

そこで、各エージェントは、それに内包されるエージェントがもつべき仕様（メソッドの名前、データ型

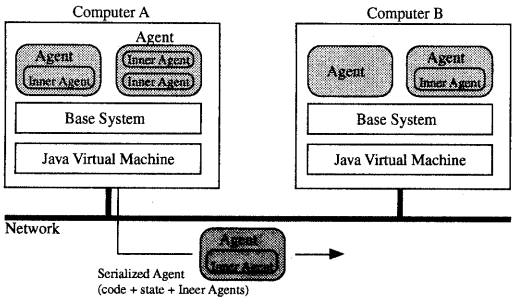

図 1: システム構成

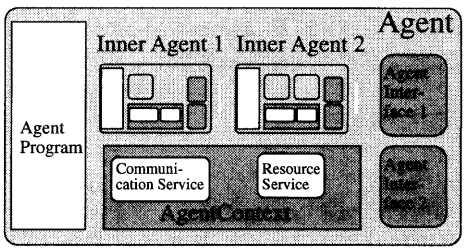

図 2: モーバイルエージェントの構成

など) を登録できるようにする。そして、エージェントが到着したときは、この仕様を満たすモーバイルエージェントだけを内包する[4]。

4 設計

4.1 システム構成

本論文で提案するシステムは、高階モーバイルエージェントを実現するシステムと、高階モーバイルエージェント作成を支援するフレームワークから構成される。これらは Sun 社 Java 言語 [1](JDK1.1 以上) により実装され、Java 言語の仮想機械により実行される。実現システムは図 1 のように構成される。

- **ベースシステム**
  各エージェントに加えて、スケジューリング、入出力、通信ネットワークなどリソースを管理・制御するプログラム。
- **高階モーバイルエージェント**
  エージェント間を移動するプログラムであり、実行スレッドが割り当てられ、能動的に動作する。また、その内部に他の高階モーバイルエージェントを含むことがある。

高階モーバイルエージェントは、同一または相違なコンピュータ上のエージェントに移動でき、また、実行状態を含めた永続化や複製が可能である。そして、このエージェントは図 2 のように構成され、その要素には次のようなものがある。

- **エージェントプログラム**
  各エージェントの動作を記述するプログラムであり、外部エージェントからの呼び出されるコールバックメソッドをもつ。また、このエージェント自体の移動・永続化・エージェント間通信要求のため、後述するエージェントコンテキストの API を呼び出しや、このエージェントが内包しているエージェントに対するコールバックメソッドの呼び出しも含まれる。
- **エージェントコンテキスト**
  内包しているエージェントに対して、エージェント間通信及びリソース情報などの各種サービスを提供するプログラムであり、メッセージ配送のためのプログラムやメッセージ待ち行列をもつことがあり、内包エージェントから通信要求を受け取ると、メッセージ配送・管理を行い、受信側エージェントにメッセージ到着を通知する。
- **内包可能エージェントの仕様**
  このエージェントに内包可能なエージェントが保持すべきメソッドの名前、返値型、引数型を規定した仕様であり、Java 言語のインタフェース型プログラムとして記述され、所定の API(`void addAbsorbableAmbient(Class spec)`) により動的にエージェントに登録される。
- **内包エージェント**
  このエージェントに内包されている 0 個以上の高階エージェントであり、エージェントの自律的な移動により入れ替わる可能性がある。また、これらの内部にさらに高階エージェントを含むことがある。

エージェントプログラムは `Ambient` クラス (図 3) のサブクラス、エージェントコンテキストは `AmbientContext` クラスのサブクラスであり、各エージェントにそれぞれ一個ずつ存在する。ただし、これらはエージェントごとにユーザによって定義することが

[4]本論文ではエージェントによる計算資源の不当利用・消費や、認証は扱わない。

できる[5]。この結果、エージェントによって内包エージェント間の通信方式やサービス内容をかえることができる。一方、内包可能エージェントの仕様と内包エージェントは複数個存在してもよい。この他、各エージェントは識別子(AmbientID)などの付加情報をもち、エージェントの内包構造はエージェントを構成するオブジェクトの参照関係として維持・変更される。

```
 1  public class Ambient {
 2  /*Constructor */
 3    public Ambient(){..}
 4  /* getting the context of this Ambient */
 5    protected AmbientContenxt getContext(){..}
 6  /* Registering an acceptable ambient spec. */
 7    protected AmbientSpec
 8    addAbsorbableAmbient(Class c){..}
 9  /* Unregistering the ambient spec. */
10    protected void
11    removeAbsorbableAmbient(AmbientSpec as){..}
12    ...
13  }
```

図3: Ambient クラス(エージェントの基本クラス)

なお、エージェントコンテキストは内包エージェントに対してAPIを通じてサービス・通信要求を受け付ける。ただし、未知の要求であった場合は、さらに外部のエージェントのエージェントコンテキスト委譲される。そして、該当する外部エージェントコンテキストが存在しないときは、ベースシステムに転送される。なお、ベースシステムはエージェント制御やサービスの基礎となるAPIとして次のようなものをもっている。

```
// エージェントの生成
    AmbientID create(byte[] data)
// エージェントの終了
    void destroy(AmbientID aid)
// エージェントの永続化
    void suspend(AmbientID aid)
// 永続化エージェントの活性化
    AmbientID resume(byte[] data)
// url参照先エージェントに移動
    void move(URL url, AmbientID aid)
// エージェントの複製
    AmbientID copy(AmbientID aid)
```

なお、ベースシステムは移動性をもたない高階エージェントであり、それが直接内包するエージェントが、移動、永続化、終了、複製する前後には、所定のコールバックメソッドを呼び出す。

ここで、相違なコンピュータにわたるエージェント移動過程を説明していく。エージェントの移動は、そのエージェントを内包する外部エージェントの移動に伴って行われる場合と、そのエージェント自身の移動要求による場合の二つがある。後者の場合は、それの外部エージェントが提供するエージェントコンテキストに用意された移動要求用のAPIを呼び出すことにより行われる。

1. 移動前に、移動通知用のコールバックメソッドを呼び出す場合には、そのメソッドを実行する。また、必要に応じて、未受理の通信メッセージの破棄・通知などの処理を行う。
2. 移動対象となるエージェントのエージェントプログラム、エージェントコンテキスト、内包可能なエージェント仕様のそれぞれのコードと、エージェントの状態、つまり、エージェントプログラムとエージェントコンテキストの状態をデータ列化する。ここで、エージェントコンテキストの状態も保存するのは、未処理のメッセージや計算環境も移動先に転送し、移動先でそのまま継続処理するためである。
3. 内部に他のエージェントを内包している場合は、内包エージェントについても同様にデータ列化する。
4. 移動エージェントを外部エージェントの内包関係リストから除き、さらに移動エージェントと内包エージェントをベースシステムの制御管理対象から外してエージェントを停止させる。
5. データ列を圧縮・暗号化した後に、移動先エージェントに転送する。

一方、新しいエージェントが到着したとき、到着先のエージェントは次の処理を行う。

1. 到着したデータ列が圧縮・暗号化されている場合は解凍・解読する。
2. データ列からエージェントプログラムに含まれるメソッドの名前・引数型・返値型を抽出する。
3. 内包可能エージェントの仕様を満足しているか判定するために、仕様中のメソッド m_i と同じ名前のメソッドが到着したエージェントに含まれるかを調べ、さらにそれの引数及び返値のデータ型を比較する。判定条件は以下の通り。$C \leq C'$ がデータ型 C は C' のサブクラスであることを表すとき、仕様側の m_i メソッドのデータ型を m_i :

[5]ユーザがエージェントコンテキストを定義するには、外部エージェントやシステムの内部情報を把握する必要があるが、これらはAmbientContextクラスにより提供され、ユーザはメッセージ配送方法などの差分を記述するだけで十分である。

$(A_i^1 \times \cdots A_i^n) \times \rightarrow A_i$とし ($A_i^j$は$j$番目引数の型、$A_i$は返値型)、到着エージェント側の$m_i$メソッドのデータ型を$m_i : (B_i^1 \times \cdots B_i^n) \times \rightarrow B_i$において

$$A_i \geq B_i \text{ かつ } B \leq A$$

となる場合は、その到着エージェントを内包可能エージェントとして判定される。一方、内包不可能と判定した場合はその旨を送信元に通知する。

4. 受信したデータ列から、エージェントプログラム、エージェントコンテキスト、内包可能なエージェント仕様などを実体化する。そして、エージェントプログラム・エージェントコンテキストを移動前状態から継続実行する。
5. 到着エージェントが他のエージェントを内包している場合は、それらの内包エージェントも同時に実体化し、それのエージェントプログラムの実行を継続する。
6. 内包関係リストに到着エージェントを登録する。また、その到着エージェントとそれに内包されているエージェントをベースシステムの制御対象に加える。
7. 到着エージェントに対して、到着通知のためにコールバックメソッドを呼び出す場合には、対応するメソッドを実行する。

同一コンピュータ上のエージェント間移動では、移動対象となるエージェントのデータ化は行わず、参照関係だけを更新する。

4.2 記述例

モーバイルエージェントの記述例を図4に示す。クラス MobilePlace のインスタンスであるエージェントは、移動する直前にメソッド leave() を、直後にはメソッド arrive() を呼び出すエージェントに内包されているとする。また、3行目より HelloStub インタフェースで示される仕様をもつエージェントを内包可能エージェントして登録する。このため、図6のエージェントが到着したときは図5の仕様を満足することから、それを内部に取り込めることになる。

```
1  public class MobilePlace extends Ambient {
2    public MobilePlace() {
3      addAbsorbableAmbient(
4        Class.forName("HelloStub"));
5    }
6    public void arrive(URL url) {
7      AmbientContext ac = getContext();
8      Enumeration e=ac.getAbsorbedAmbients();
9      while(e.hasMoreElements()) {
10       ac.service("send",
11       (AmbientID)e.nextElement(), "resume");
12     }
13   }
14   public void leave(URL url) {
15     AmbientContext ac = getContext();
16     Enumeration e=ac.getAbsorbedAmbients();
17     while(e.hasMoreElements()) {
18       ac.service("send",
19       (AmbientID)e.nextElement(), "suspend");
20     }
21   }
22 }
```

図4: エージェントプログラムの例

```
1  interface HelloSpec extends AmbientSpec {
2    public void suspend();
3    public void resume();
4    public void hello(String message);
5  }
```

図5: 内包可能エージェントの仕様

5 実装・評価

ここではベースシステムを中心に、実装について概観する。エージェントの移動において、移動対象となる情報はエージェントの実行状態やプログラムコードに加えて、識別子などの付加情報である。また、そのエージェントが内部に他のエージェントをもつ場合は、それらのエージェントの実行状態、コード、内包関係も移動対象に含める。なお、エージェントの実行状態はベースシステムを通じてバイト列に変換される。バイト列は移動先コンピュータに転送する前に転送速度の向上のため実行状態及びクラスファイルは ZIP 形式で圧縮される。転送は TCP/IP 上で実現された HTTP の拡張通信プロトコルを利用する。

なお、エージェントの実行状態は JDK1.1 の直列化機構を利用してバイト列化するが、JDK1.1 の仮想機械上の制約によりヒープ領域上の実行状態のみがその対象となり、メソッド実行中に利用するスタック領域内の情報は対象にならない。この結果、移動や永続化において実行中メソッドのローカル変数や実行カウンタは保持されない。また、エージェントはその内部で複数の実行スレッドを利用できるが、移動・永続化の前にスレッドの停止処理を明示的に与える必要がある。

到着したエージェントの内包可能性の判定は、Java言語のイントロスペクション機能により実現する。

次に、性能評価を示す。例えば、エージェントが他のエージェントへ移動するのに要する時間(秒)は下表のようになる。

表：エージェントの移動速度

相違なコンピュータ間の移動	0.6s
同一コンピュータ上の移動	0.1s

IBM-PC/AT 互換機 (PentiumPRO:200MHz)x2 の Ethernet 接続 (10BASE-T)

既存モーバイルシステムの多くは、エージェントの転送方法として、Java言語のRMI機構、または、HTTPと同様なリクエスト・レスポンス形式を利用する。前者はネームバインディングのコストが大きく、一方、後者ではオンデマンドにクラスファイルなどの必要情報を転送するために、リクエスト・レスポンス回数ごとにTCP接続・切断を繰り返す必要がある。これに対し、本システムでは一回のTCP接続でエージェントを転送するため、エージェントを構成する情報・ファイルの数が多いほど相対的に移動速度が速くなる。さらに、携帯端末など通信回線の不調・切断が繰り返される分散システムにも対応できるという特徴をもつ。

```
 1  public class Hello extends Ambient
 2      implements HelloStub {
 3  // aid is the ID of a MeetingPlace Ambient
 4    public Hello() {
 5      ....
 6      ac.in("matp://some.where.ac.jp/"+aid);
 7    }
 8    public void suspend() {
 9      ....
10      ac.out("matp://some.where.ac.jp"+aid);
11    }
12    public void resume() {
13      AmbientContext ac = getContext();
14      Enumeration e=ac.getColleagueAmbients();
15      while(e.hasMoreElements()) {
16        ac.service("send",
17          (AmbientID)e.nextElement(),
18            "hello", "How are you?");
19      }
20    }
21    public void hello(String message) {....}
22  }
```

図6: 到着エージェントのエージェントプログラム

6 関連研究

エージェントやプロセスなどの計算実体を移動する方法には、OSにより実現するもの(例、DEMOS/MP [16], Apertos [24]など)や、プログラミング言語として実現するもの(例、Emerald [3]、SOS [20]など)がある。また、モーバイルエージェントと称されるシステムには、Telescript [22]、Safe-Tcl [15]、Agent-Tcl [6]、そしてJava言語をベースにしたものには、IBM社のAglets [8]、ObjectStore社のVoyager [14]、General Magic社のOdyssey [5]、Mitsubishi Electric社のConcordia [23]、Shuttgart大学のMole [21]などがある。ただし、これらの既存のモーバイルエージェントシステムにおいて、モーバイルエージェント内部に別のモーバイルエージェントを入れ子状に保持できるものはない。また、プレースと呼ばれる概念により複数のモーバイルエージェントを含有可能な計算実体を提供するものがあるが、プレースは移動能力のないステーショナリエージェントである。また、Moleに対してエージェントのグループ化機能を導入したシステム [2] があるが、グループ内モーバイルエージェントに対する多様なメッセージ配送を目的とし、グループ自体の移動性は提供しない。また、高階オブジェクトを考慮したRPCやオブジェクト転送方法が幾つか提案されている [10, 13, 12]。しかし、高階化されたオブジェクトそれぞれが独立に能動性をもつことは考慮されておらず、モーバイルエージェントの転送技法には不向きである。

7 まとめ

この論文では、Mobile Ambientsに基づきながら、モーバイルエージェントの合成・階層化を実現できるモーバイルエージェントシステムの設計・評価を行った[6]。また、このシステムはエージェント間通信やエージェント制御方法を動的に変更できるため、柔軟なシステム構築を可能にするという特徴をもっている。なお、モーバイルエージェントの階層的合成は実用面からも重要である。例えば、複数のモーバイルエージェント合成体から構成されたシステムは、モーバイルエージェントの移動性により、システムを停止することなく動的にシステム構成を変更できるようになり、オンラインバージョンアップや進化的分散計算システムの実現に道を開くものとなる。

[6]本システムのソースプログラム及びバイナリは次のURL http://chopin.is.ocha.ac.jp/agent からダウンロードできる(Java言語JDK1.1以上が動作する計算システムで稼働)。ただし、本論文執筆時(1998年3月)において配布されているシステムは一階のエージェントのみ対応。

最後に今後の課題を述べる。現行のシステムでは、エージェント移動において移動先のエージェントを特定する必要があり、移動先の相対的な特定や、移動先エージェントの階層変化に柔軟に対応できない。このため、エージェント移動経路及び移動先の動的な選択機構が必要となる。また、モーバイルエージェントでは、移動先ホスト（本システムではエージェント）のリソース保護やセキュリティが重要となるが、本システムでは十分に対処できるとはいえず、セキュリティ・保護機構を導入することは重要な課題となる。

モーバイルエージェントの移動性は、エージェントの位置の変更だけでなく、移動時に生じる計算の不連続性や、計算資源の不当利用など従来の分散計算システムではない不確定性や障害を含む。このため、正当かつ効率的なシステム構築には、機能的な動作内容だけでなく、位置概念や時間などの多様なシステム特性を扱える理論的な枠組み (例えば [17, 18, 19]) による仕様記述や解析が重要となる。

参考文献

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

[2] J.Baumann and N.Radounklis, *Agent Groups in Mobile Agent Systems*, Conference on Distributed Applications and Interoperable Systems, 1997.

[3] A.Black, N.Hutchinson, E.Jul, and H.Levy, *Object Structure in the Emerald System*, Proceedings of ACM Conference on Object Oriented Programming Systems, Languages, and Applications, pp.78-86, 1986.

[4] L.Cardelli and A.D.Gordon, *Mobile Ambients*, to appear in Proceedings of European Joint Conference on Theory and Practice of Software, March, 1998.

[5] General Magic, Inc. *Introduction to the Odyssey*, http://www.genmagic.com/agents, 1997.

[6] R.S.Gray, *Agent Tcl: A Tranportable Agent System*, Proceedings of Workshop on Intelligent Information Agents, 1995.

[7] C.G.Harrison, D.M.Chess, and A.Kershenbaum, *Mobile Agents: Are they a good idea?*, Research Report, IBM Research Division, T.J.Watson Reserach Center, March 1995.

[8] IBM Tokyo Reserach Laboratory, *Aglets Workbench: Programming Mobile Agents in Java*, http://www.trl.ibm.co/aglets, 1997.

[9] E.Jul, H.Levy, N.Hutchison, and F.Schneider, *Fine-Grained Mobility in the Emerald System*, ACM Transaction on Computer Systems, Vol.6, No.1, pp.109-133. 1988.

[10] F.C.Knabe, *Language Support for Mobile Agents*, Technical Report ECRC-95-36, European Computer-Industory Research Centre, 1995.

[11] Milner, R., Parrow. J., and Walker, D., *A Calculus of Mobile Processes*, Information and Computation, Vol.100, pp.1-77, 1992.

[12] M.Mira da Silva, *Models of Higher Order, Type Safe, Distributed Computation over Autonomous Persisitent Object Stores*, PhD thesis, University of Glasgow, 1996.

[13] A.Ohori and K.Kato, *Semantics for Communication Primitives in a Polymorphic Language*, Proceedings of ACM Symposium on Principles of Programming Languages, pp.99-112, 1993.

[14] ObjectSpace Inc, *ObjectSpace Voyager Technical Overview*, ObjectSpace, Inc. 1997.

[15] J.K.Ousterhout, J.Levy, and B.Welch, *The Safe-Tcl Security Model*, Sun Microsystems Laboratory, 1996.

[16] M.L.Powell and B.P.Miller, *Process Migration in DEMOS/MP*, ACM Symposium on Operating Systems Principles, pp.509-514, 1983.

[17] I.Satoh and M.Tokoro, *A Formalism for Real-Time Concurrent Object-Oriented Computing*, Proceedings of ACM Conference on Object Oriented Programming Systems, Languages, and Applications, pp.315-326, 1992.

[18] I.Satoh and M.Tokoro., *Time and Asynchrony in Interactions among Distributed Objects*, Proceedings of European Conference on Object Oriented Programming, LNCS 952, pp.331-350, Springer-Verlag, 1995.

[19] I.Satoh, *An Algebraic Framework for Optimizing Parallel Programs*, to appear in Proceedings of Symposium on Software Engineering for Parallel and Distributed Systems, IEEE Press, April, 1998.

[20] M.Shaprio, P.Gautron, and L.Mosseri, *Persistence and Migration for C++ Objects*, Proceedings of European Conference on Object Oriented Programming, pp.191-204, 1989.

[21] M.Strasser and J.Baumann, and F.Hole, *Mole: A Java Based Mobile Agent System*, Proceedings of ECOOP Workshop on Mobile Objects, 1996.

[22] J.E.White, *Telescript Technology: Mobile Agents*, General Magic, 1995.

[23] D.Wong, N.Paciorek, T.Walsh, *Concordia: An Infrastructure for Collaborating Mobile Agents*, Proceedings of Workshop on Mobile Agents'97, LNCS Vol. 1219, 1997.

[24] Y.Yokote, The Apertos Reflective Operating System: The Concept and its Implementation, Proceedings of ACM Conference on Object Oriented Programming Systems, Languages, and Applications, pp.414-434, 1992.