エージェント指向言語 MALS
Agent Oriented Language, MALS

勝野 恭治　　**中原 真則**　　**所 眞理雄**
Yasuharu KATSUNO　　Masanori NAKAHARA　　Mario TOKORO

慶應義塾大学大学院 計算機科学専攻
Department of Computer Science, Keio University

概要

既存のモーバイルエージェント記述言語は、複数の手続きを全て実行するか全く実行しないという、手続きの原子的実行が保証されていなかった。

本論文では、言語レベルで手続きの原子的実行を保証しているエージェント指向言語 MALS(Mobile Agent Language System) の提案、実装を行う。MALS を用いることによりプログラマは、プログラム実行中の障害を意識する必要がなくなる。このためプログラマの負荷を軽減することができる。

1　はじめに

近年、World Wide Web 上における情報検索など、広域に分散した計算機間にわたる作業が増加している。また、無線ネットワークを用いてネットワークに接続でき、どんな場所にいても様々なサービスを受けられるようになった。

このような場合のプログラミングには、従来 RPC が用いられていた。しかし RPC では、広域に分散した作業を行う際に、次の 2 点が問題になる。

(1) 1 往復の通信ごとに行なわれる計算が、手続きという小さい単位になっている。そのため、複数の手続きを実行し終わるまでの時間が長い。

(2) ネットワークの接続状況、サーバの位置などは絶えず変化する。RPC では、プログラマがプログラム実行時のネットワーク状況を把握できない。よってこの変化に対処することができない。

また、無線ネットワーク等を用いてモーバイルコンピュータを広域ネットワークに接続する際に、RPC を用いると次の点が問題になる。

(3) 手続き呼出しを行ってから返答が返ってくるまで、ホスト間を接続し続けなければならない。そのため、モーバイルコンピュータをネットワークに接続し続ける時間が長くなる。

本論文では RPC の問題点を克服するために、プログラムを目的のホストに移動させ実行させるリモートプログラミング [6] [7] が登場したと考えている。リモートプログラミングを用いると遠隔手続き呼出しに用いられるプログラムの単位が RPC に比べて大きいのでホスト間の通信を減らせ、これにより遅延が大きいネットワークを介した通信の回数も減らせる。そのため、(1) を解決することができる。また、プログラムがホスト間を移動する間のみネットワークと接続すればよいので (3) を解決することができる。しかし RPC の場合と同じく、プログラマがプログラム実行時のネットワーク状況を把握できないので、(2) が問題となってしまう。

そして本論文では、RPC、リモートプログラミングの問題点を克服するためにモーバイルエージェント [3] [4] が登場したと考えている。モーバイルエージェントの定義は論文によって異なるが [3] [4]、本論文ではモーバイルエージェントを、『ネットワーク上を自律的に移動し、各計算機上に存在する情報を収集、利用しながらプログラマの要求を満たすプログラム』と定義する。モーバイルエージェントを用いることによって、リモートプログラミングの場合と同様に (1)(3) を解決することができる。そしてモーバイルエージェントは、各計算機上から情報を収集、利用しモーバイルエージェントの着目している局所的なネットワーク状況を把握できるので (2) を解決することができる。

しかし既存のモーバイルエージェント記述言語 [3] [4] では、手続きの原子的実行が保証されていなかった。本論文では、複数の手続きが全て実行されるか全く実行されないという特性を手続きの原子的実行と定義する。手続きの原子的実行を保証する言語を用いると、プログラマはプログラム実行中の例外処

理を意識する必要がなくなる。

本論文では、既存のモーバイルエージェント記述言語の問題点を踏まえて、手続きの原子的実行を保証するエージェント指向言語 MALS(Mobile Agent Language System) を提案し、実装を行う。

本論文は、本章も含めて 7 章から構成されている。第 2 章で、MALS のシステムモデル、第 3 章でアプリケーション例、第 4 章で MALS の実装についてそれぞれ説明する。そして、第 5 章で評価と検討を行い、第 6 章で今後の課題を述べる。最後に、第 7 章で結論を述べる。

2 MALS のシステムモデル

本章では、手続きの原子的実行、および MALS のシステムモデルについて説明する。

2.1 手続きの原子的実行

手続きの原子的実行を保証する際の障害として次の 4 つのものが考えられるが [5]、

- トランザクション障害
- システム障害
- 媒体障害
- ネットワーク障害

本論文ではトランザクション障害、つまりデータベースへのアクセスの失敗のみを考える。この障害から回復するには、原子性を保証する範囲のアクセスを取消し、データをアクセスを行う前の状態に戻さなければならない。

2.2 MALS のシステムモデル

MALS は MALS エージェント、Place エージェント、データベースによって構築されている (図 1)。

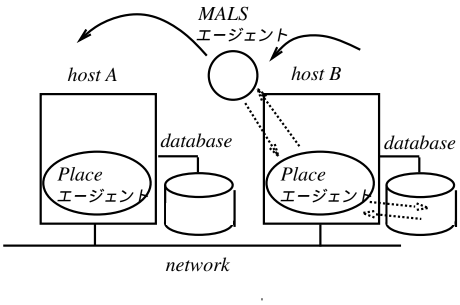

図 1　MALS を構築するシステム

2.2.1 MALS エージェント

MALS エージェントは、ホスト間を自律的に移動し、ホスト上に存在する Place エージェントと通信を行いながらプログラマの要求を満たすプログラムである。Place エージェントとの通信方法は次の 2 通りがあり、プログラマが選択する。

- Place エージェントが存在するホストまで移動して通信を行う
- ネットワークを介して通信を行う

自律的に移動するとは、各計算機上に存在する情報の収集、利用によってプログラム中に記述されていない移動を行うことを示す。例を挙げると、モーバイルエージェントは、データベースを管理するホストが、ホスト A からホスト B に変更されプログラマが知らなくても、その変更を知り、プログラム中に記述されていない、ホスト B への移動を行う。

2.2.2 Place エージェント

Place エージェントは、各ホストに 1 つずつ存在し、MALS エージェントと通信を行うことによって MALS エージェントに対してさまざまなサービスを提供する。提供するサービスを以下に示す。

- データベースを管理するホストの URL を提供
  Place エージェントは、あるデータベースを所有しているホストの URL、またはその Place エージェントがいるホストにあるデータベースに対してリンクを張っていることを知らせているホストの URL を保持している。そしてその URL を、MALS エージェントからの要求にあわせて提供する。またその URL は、Place エージェント同士通信することによって更新される。
- MALS エージェントを移動
  MALS エージェントは、Place エージェントに依頼することによって指定したホストに移動させてもらう。
- ホスト上のデータベースにアクセス
  MALS エージェントが、あるホスト上のデータベースにアクセスするにはそのホスト上の Place エージェントに依頼する。Place エージェントを通してデータベースにアクセスさせることで、MALS エージェントがデータベースに対して不当なアクセスを行うのを防ぐことができる。

3 アプリケーション例

本章では、インターネット上における MALS のアプリケーションとして、電子商取引を例に挙げる。

現在、インターネットはビジネスの分野でも用いられている。その例が電子商取引である。本や CD、また花までもがインターネット上で購入でき、今後様々な品物が購入できると考えられる。電子商取引のアプリケーションを MALS で記述すると、2 つの問題点が解決できる。

まず、モーバイルコンピュータでインターネットを用いて電子商取引を行う場合、不安定なネットワークとの接続、高額な通信料金という状況の中で長時間インターネットと接続しなければならないという

問題点がある。特に、モーバイルコンピュータではネットワークと接続する時間をできるだけ短くしたい。そこで MALS を用いると、ネットワークとの接続時間を短くすることが可能である。

次に、複数の電子商取引を原子的に実行しようとすると手間がかかってしまうという問題点がある。旅行に行くプログラマがインターネット上で飛行機のチケット、フェリーのチケット、ホテルの予約を行う状況を考える。3 つの電子商取引を行う場合、現在ではそれぞれのデータベースを所有しているホストにアクセスして予約を取らなければならない。またこの場合、予約をするなら 3 つ全て取らないと意味がない。そこで、もしどれか 1 つでも予約が取れなかった場合は、今まで予約した分を全てキャンセルしなければならず、手間がかかってしまう。しかし、MALS を用いると MALS エージェントを一度作成するだけで、予約からキャンセルまで全てが行える。

4 MALS の実装

本章では、MALS の実装方法、記述構文を述べる。

4.1 MALS の実装方法

本研究では、Sun Microsystems が開発した JDK(Java Developer's Kit) 1.0.1 [1] と、電子技術総合研究所の平野 聡氏が開発した Java の ORB(Object Request Broker) である HORB 1.2.1 [2] を用いて MALS を実装した。また、データベースは David J. Hughes 氏が開発した mSQL 1.0 を用い、Java から mSQL へのアクセスとして Darry Collins 氏が開発した mSQL for JAVA 1.1.1 を用いた。

mSQL にはトランザクション機能が付いていない。そこでデータベースに対するアクセスの制御には 2 相ロック [5] を、分散トランザクションには 2 相コミットメントプロトコル [5] を用いた。

また、本論文では図 2に示すように、MALS ソースを Java の中間コードに変換して実行する。トランスレータ mals を Perl 5.001 を用いて実装した。

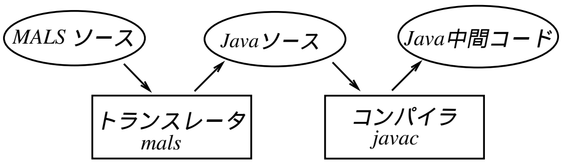

図 2 MALS ソースから Java の中間コードへの変換

4.2 MALS エージェントの記述構文

以下の実行を行う MALS エージェントの記述は図 3 のようになる。

1. ホスト aaa に移動してデータベース DB1 にアクセス
2. ホスト bbb に移動して Place エージェントからデータベース DB2 を所有しているホストの URL を獲得
3. DB2 が存在するホストに移動して DB2 にアクセス

図 3でもし aaa、host1 にそれぞれ DB1、DB2 が存在しない場合は MALS エージェントが aaa,host1 上のそれぞれの Place エージェントから DB1、DB2 を所有しているホストの URL を収集する。その URL が得られたら、それぞれのデータベースが存在するホストに移動し、目的のデータベースにアクセスする。情報が得られなければエラーとなり、プログラマが処理を行う。

```
agent A1 {
 static String host1 = new String();
 beginLocation( aaa ){
  selectDB("DB1");
  MsqlResult rs1 = Query("select * from seat");
  upDate("katsuno", "002");
 }endLocation

 beginLocation( bbb ){
  host1 = getInfomation("DB1");
 }endLocation

 beginLocation( host1 ){
  selectDB("DB2");
  MsqlResult rs1 = Query("select * from seat");
  upDate("katsuno", "004");
 }endLocation
}
```

図 3 MALS エージェントの記述例 1

データベースの更新を行わないオペレーションは mSQL for JAVA の構文をそのまま使用し、更新を行うオペレーションはデータベースの一貫性を保証するため変更してある。他の記述構文は Java、HORB に以下に説明する部分を加えた形になっている。

- **agent** *MobileAgentName*{ **.....** }
  MALS エージェントを宣言する。MALS の **agent** と Java の **class** にはメソッド、クラスの定義ができない、という点で違いがある。
- **beginLocation**(*String HorbURL*) { **.....** }**endLocation**
  HORB で定義されている *HorbURL* に該当するホストに移動を行い、中括弧でくくられている手続きを実行する。
- **String getInformation**(*String DBName*);
  Place エージェントから *DBName* を所有しているホストの HorbURL を得る。*DBName* の管理は、本論文の時点では単一サーバで管理している。しかし今後は、*DBName* で木構造を構築し、DNS のように管理を行う。

- **beginAtomic { ..... }endAtomic**
  中括弧でくくられた部分の手続きに対する原子的実行を保証する。

MALSエージェント、Placeエージェント間のネットワークを介する通信はHORBの構文を用いる。

5 評価および検討

5.1 手続きの原子的実行に関する記述性

先程の図3で示したプログラムは手続きの原子的実行を保証していない。これを保証するように書き換えると図4のような記述になる。

```
 agent A1 {
  static String host1 = new String();
   beginAtomic{
    beginLocation( aaa ){
      selectDB("DB1");
      MsqlResult r1=Query("select * from seat");
      upDate("katsuno", "002");
    }endLocation

    beginLocation( bbb ){
      host1 = getInfomation("DB1");
    }endLocation

    beginLocation( host1 ){
      selectDB("DB2");
      MsqlResult r2=Query("select * from seat");
      upDate("katsuno", "004");
    }endLocation
   }endAtomic
}
```

図4 MALSエージェントの記述例2

図4を図3と比較すると、ほぼ変わらぬ記述で手続きの原子的実行が表現できている。すなわち、手続きの原子的実行を保証する際のプログラマの負担は小さいと考えられる。

5.2 手続きの原子的実行に関するオーバーヘッド

図3、図4それぞれの実行時間を測定し、手続きの原子的実行を保証する際のオーバーヘッドを評価する。測定は、ホスト`aaa,bbb,host1`全てが同一ホストである場合について行った。実行環境は、PC/AT互換機(Intel Pentium120MHz,RAM 32M)上でLinuxを起動し、図3、図4それぞれを5回実行させ、1msの数値を四捨五入したものの平均を取った。表1に測定結果を示した。

図3	図4
930 ms	1206 ms

表1 手続きの原子的実行に関するオーバーヘッド

手続きの原子的実行に関する全体のオーバーヘッドを表1から求めると29.68%となった。

オーバーヘッドが大きい要因の一つとして、分散トランザクションを行う際に用いた2相コミットメントプロトコルが考えられる。よって、分散トランザクション処理の手法を変えればオーバーヘッドがいくぶん減少するのではないかと考えられる。

6 今後の課題

6.1 モーバイルエージェントの特性を生かしたトランザクション処理手法

本論文では、トランザクション処理手法として2相ロックと2相コミットメントプロトコルを用いたが、共にモーバイルエージェントの特性を生かしたトランザクション処理手法とは考えにくい。今後、MALSをモーバイルエージェントの特性を生かしたトランザクション処理手法で実装する必要がある。

6.2 セキュリティ

MALSでは、不特定多数のMALSエージェントがホストに移動してくる。そのため、ホストに害を与えるMALSエージェントの移動も当然考えられる。本論文の時点ではホストに害を与えるMALSエージェントに対する対処が行われていない。そのため実運用の際に考慮せねばならない。

7 結論

既存のモーバイルエージェント記述言語には、手続きの原子的実行を保証するものは存在していない。

そこで本論文では、エージェント指向言語MALS(Mobile Agent Language System)の提案、実装を行った。MALSでは、手続きの原子的実行を言語レベルで保証している。よって、プログラマは、プログラム実行中の障害を意識する必要がなくなる。このためMALSは、プログラマの負荷を軽減できる。

本論文ではさらに、MALSの記述性に関する評価を行ない、複数の操作の原子的実行を保証する場合におけるMALSの記述性の高さを示した。

参考文献

[1] Gosling, J. and McGliton, H. The Java language environment: A White Paper. *Sun Microsystems.* 1995.

[2] 平野 聡. HORB: ワールドプログラミングのための並行分散オブジェクト指向言語. In *Workshop on Object Oriented Conference '96(WOOC '96)* , 1996.

[3] James E. White, Bradshaw, Jeffrey(ed.). Mobile Agents. *AAAI Press and MIT Press*, 1996.

[4] Markus StraBer, Joachim Baumann, Fritz Hohl. Mole - A Java Based Mobile Agent System. In *ECOOP '96 Workshop on Mobile Object Systems*, 1996.

[5] P. Bernstein, V. Hadzilacos, and N. Goodman. Concurrency Control and Recovery in Database Systems. *Addison Wesley*, 1987.

[6] Andrew Black, Norman Hutchinson, Eric Jul, and Henry Levy. Object Structure in the Emerald System. In *OOPSLA '86 Proceedings*, Sep, 1986.

[7] Luca Cardelli. A Language with Distributed Scope. In *Computing Systems*,vol.8,No.1,MIT Press, 1995.