B-034

# モバイルエージェントシステム AgentSphere の開発 — エージェントへの移動指示を行う API の設計と実装 —

# Development of Strong Migration Mobile Agent System AgentSphere - Design and Implementation of an API to direct Mobile Agent to migrate -

大久保 秀† 佐藤 寿‡ 甲斐 宗徳†
Shu Okubo Hisashi Sato Munenori Kai

## 1. はじめに

FIT2012 で報告した動的なオブジェクト操作が可能なシェル[1]により，AgentSphere を停止させることなく，エージェントやモジュールの操作が可能となり，エラーを起こした問題点の特定や修正を動的に行えるようになった．これによりエージェントの生成・起動・変更については実行中にユーザが動的に行えるが，エージェントの移動はエージェント自身の自律性に任されており，移動先のマシン，移動するタイミングを外部から指定することは不可能だった．しかし，自律的なエージェントの移動だけでは対応できない様々な状況が想定されるため，実行中のエージェントの移動を外部から制御することも必要である．例えば，あるマシンでエージェントを大量に生成したいとする．これを一定の時間内で行うと，生成された各エージェントが自律的に移動を行う前に，大量のエージェントが活動することによってそのマシンに大きな負荷がかかってしまう場合がある．そこで，エージェントが自律的に移動する以外に，人為的にそのエージェントに他のマシンに移動するよう指示することによって，自分のマシンで活動を行うエージェントの数をユーザが操作することができ，負荷をかけず安全にエージェントの生成を行うことができる．

そこで本年度は，エージェントの移動制御を行うための API を作成し，その API をシェルから利用するためのシェルコマンドを準備した．

## 2. シェルによるエージェント移動操作の方法

AgentSphere はエージェントが自律的に移動するきっかけとして，マシンにかかる負荷の推定値を計算し，ある一定の値を超えたときに移動するという仕組みをとっている．この判断基準の値は，マシンにかかる負荷によって JVM の処理能力が落ちない最大の値が望ましい．そこで，その値をソースコード上で静的に定義し AgentSphere を起動すると，負荷の推定値が判断基準の値に達するまで，エージェントが AgentSphere 内に入ってくる．しかし，判断基準の値が大きすぎる場合，エージェントが AgentSphere 内にいることによって負荷が上がり JVM の処理能力が落ちたとしても，負荷の推定値が判断基準の値に達していないために，エージェントが自律して移動しないという状況が発生する．そうなったときに，シェルからエージェントに人為的に移動指示を出し，JVM の処理能力が落ちない状態になるまでエージェントをいくつか他のマシンへ移動させれば，負荷分散を効率的に行うことができる．また，JVM の処理能力が落ちなくなるそのときの負荷の推定値こそ，判断基準の値と見積もることができる．

AgentSphere の実行環境である JVM は，プログラムカウンタの値を取得し，それを利用してプログラムの途中から実行するということが不可能である．そのため，エージェントの実行中にシェルから移動指示を出しても，それを認識させ，指示が出されると同時に他のマシンへの移動を行い，移動先で支持が出されたソースコード上の箇所から活動を再開するということは困難である．動作中の Java のプロセスを他のマシンへ移動させるためには，例えば，コード中に移動指示があるかどうかというのをリアルタイムで常にチェックする仕組みを作るという方法がある．しかし，これでは動いているプロセスを監視するために別プロセスを常に動かし続ける必要があり，余計なリソースを消費してしまう．また，仮にこれが可能だとしても，移動のためにシリアライズされるエージェントが，どのような実行状態でシリアライズされるかがユーザにはわからない．つまり，ソースコード上のスコープの深いところや，再帰関数の深いところといったスタックを多く使用している状態でシリアライズを行う可能性がある．この場合，大量のデータを移動させることになるため，オーバヘッドも大きくなってしまう．これを解決するためには，エージェント移動時の持ち運びデータが必要最小限となるよう，スタック使用量が少ないソースコード上の箇所で移動させる必要がある．

そこで，実行中のエージェントに移動指示を出すのではなく，AgentSphere に備え付けられた自己バックアップ機構によって定期的に作成されるエージェントのバックアップに移動指示を出すことにした．移動したエージェントのバックアップは移動先のマシンで処理を再開する．一方，自分のマシンで実行中のオリジナルのエージェントには，その処理を停止させる．これにより，移動と同様の振る舞いをさせることができる．そこで，この一連の動作を行わせるための API を作成し，これをシェルで move コマンドを実行することによって呼び出せるようにした．また，コマンド実行時に移動指示を出すエージェントと移動先の IP アドレスを指定することによって，エージェントの移動操作を実現した．

## 3. エージェントの自己バックアップ機構の概要

分散処理では，複数のマシンに処理を分散させることによって信頼性や耐障害性を高めることができる．しかし，故障や停電などの不測の事態によりマシンがダウンしてしまうと，それまでマシンで行っていた処理が無駄になってしまう．そこで，処理がある程度進んだときにエージェント自身のバックアップを取ることによって，障害が発生したときにバックアップを取った地点からの再開が可能になる．このバックアップとはエージェント自身のインスタンスをシリアライズし，作成されたバイナリデータのことである．バックアップの作成は，エージェントのソースコード内に自動で複数挿入される backup メソッドを実行したときに行われる．この backup メソッドを挿入する箇所は，バックアップによるオーバヘッドをできるだけ少なくするために，1 回のバックアップに関わる実行時データができ

†成蹊大学理工学研究科理工学専攻 Graduate School of Science and Technology, Seikei University
‡成蹊大学理工学部情報化学科 Department of Computer and Information Science, Seikei University

るだけ少なくて済む箇所が望ましい．例えば，スコープが深い箇所ほど，スタックフレーム数が増え，取得する情報量が大きくなってしまい，それらをバックアップして持ち運ぶのは，オーバヘッドが大きくなる．そこで，スコープから出た位置やメソッド呼び出しの直後が backup メソッドの挿入位置となっている．過去の研究で，この自己バックアップ機構が実装された[2]．しかし，この機構で作成されたバックアップはユーザが人為的に違うマシンへと移動させることができなかったため，この機構を再構築し，ユーザの任意で移動させられるバックアップを作成できるようにした．

### 3.1 バックアップを利用したエージェントの移動

バックアップはエージェントの処理の最中に自動的に作成される．ユーザがシェルを介してエージェントへの移動指示を出したとき，自分のマシンに作られたバックアップを移動先のマシンに送り，そこで復元させることによって移動と同様の振る舞いをさせるようにした．

バックアップからの復元は，負荷が大きくなったマシンでエージェントが処理し続けるのが困難になったときや，オリジナルのエージェントが不測の事態により失われたとき，実行していたマシンとは別のマシンに存在するそのエージェントのバックアップを対象に行われる．しかし，今回はエージェントが移動したように見せかけるため，復元させるバックアップは自分のマシンに存在するバックアップが対象であり，それが他のマシンへ移動したと同時に復元されるようにした．

一方，オリジナルのエージェントについて，通常はバックアップからの復元を行われる段階ですでに処理を続けるのが困難か，失われている状態になっている．しかし，移動に見せかける場合，バックアップを他のマシンへ移動させ復元してもオリジナルのエージェントは自分のマシンで動作中であるため，同じエージェントが 2 つ動いているという状況が発生してしまう．これを防ぐために，バックアップ復元と同時に自分のマシンで動作中のオリジナルのエージェントを破棄する必要がある．そこで，backup メソッドの返り値を bool 型にし，バックアップ移動後にオリジナルのエージェントが backup メソッドを実行したときに，backup メソッド内で新たなバックアップを作成せず，エージェントが処理を停止するためのフラグを返すようにした．

図 1 にバックアップを利用したエージェントの移動方法を示す．図中のソースコードを指している矢印はプログラムカウンタの位置を表している．図では処理群 C を実行している最中にシェルから移動指示を出している．処理群 B が終了した直後のバックアップが最新のバックアップとしてマシン A に存在するため，これをマシン B に送り，移動先で復元して再開を行う．

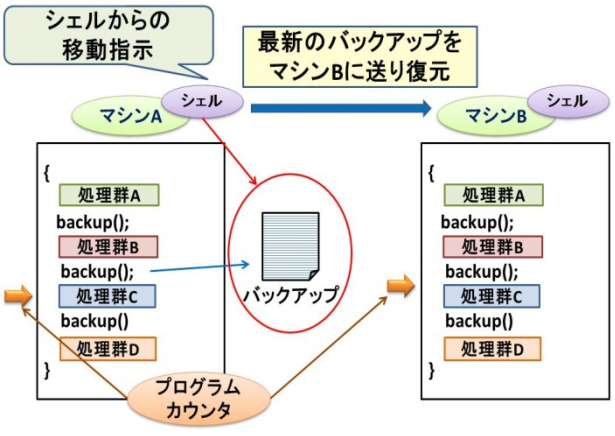


**図 1.バックアップを利用したエージェント移動**

### 3.2 バックアップを行ったソースコード上の箇所からの再開方法

Java 環境では，スタック領域の変数やプログラムカウンタの値をシリアライズすることができない．そこで，バックアップ時にスタック領域の変数を一時的にヒープ領域に退避させることでスタック領域内の変数のシリアライズを実現している．また，エージェントのソースコードを if-else 文を使用してバックアップ前とバックアップ後の処理に分割し，バックアップの復元時には backup メソッドまでの処理を飛ばすようにすることでプログラムカウンタの代わりとしている．ここで，途中から再開をするためのフラグを BackupFlag と定義している．

この変換は，エージェントを移動するためのメソッドである migrate が含まれているときのソースコード変換とほとんど同じ構造をしている．

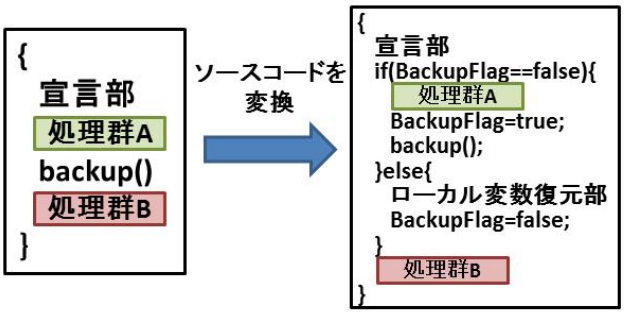


**図 2.バックアップを含むソースコードの変換**

### 3.3 migrate メソッドと backup メソッドの実行後の振る舞いの相違点

migrate メソッドと backup メソッドは，実行されたときにエージェントがマシンを移動するかしないかの違いがある．migrate メソッドが実行されたときは，必ずソースコードの先頭からの実行となるが，backup メソッドの場合は移動が発生しないため，ソースコードの先頭に戻るということはない．この違いによってソースコードの変換方法にも違いが起こる．過去の研究[3]で，ループ文の中で migrate メソッドが呼び出されたときの変換方法が示された．図 3 にその一例を示す．

ループ文の内部で migrate メソッドが実行された場合，移動先マシンでは無条件にループの中に入る必要がある．そのため，条件文に「MigFlag==true」という条件を追加し，元の条件文との論理和を取ることでこれを実現している．

backup メソッドの存在する部分にも同様にソースコードの変換を行うと，振る舞いの違いにより，1 周目のループでフラグが true となり，2 周目のループでは必ずループに入ってしまう上に処理群 B が飛ばされてしまうという問題が起こる．そのため，ループ文の中に backup メソッドが存在する場合のソースコード変換では，図 4 のように backup メソッドが実行された後にフラグの値を false に戻す必要がある．

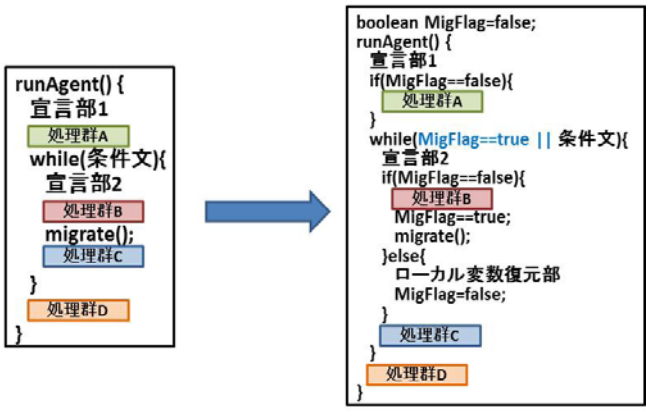


**図 3.ループ文の中に migrate メソッドがある場合の変換**

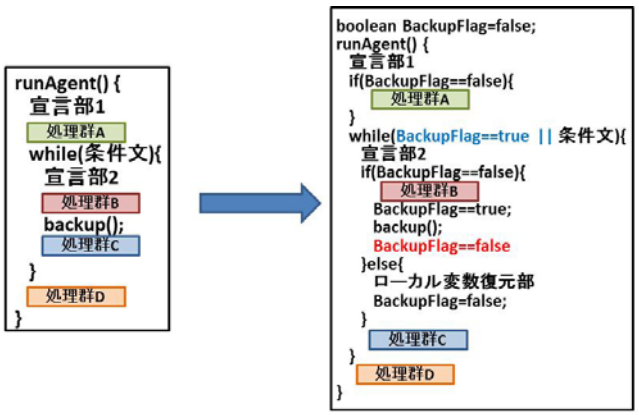


**図 4.ループ文の中に backup メソッドがある場合の変換**

### 3.4 バックアップファイルの保存

backup メソッドは，int 型の変数である BACKUPNUM の初期値を 0 として，以下のように引数を持つ．

backup(this, ++BACKUPNUM, continueFlag)

第 1 引数は，エージェント自身のインスタンスである．これを backup メソッドに渡して，メソッド内でシリアライズを行う．第 2 引数は，バックアップのバージョンを表す番号である．インクリメントをしているため，backup メソッドが実行される度に値が大きくなる．数字が大きいほど新しいバックアップであるということを表す．新しいバックアップが作られたら，そのマシンに存在する古いバックアップは削除される．バックアップファイルは，

(エージェント固有の ID)_(バックアップ番号).bak

という名称で保存される． 第 3 引数は，バックアップを作成するか，しないかを判定する bool 型のフラグである．このフラグが false だった場合，新たなバックアップは作成されず，そのときの backup メソッドで行う処理は，そのマシンに存在するバックアップの削除のみである．このフラグが false になる場合とは，バックアップが他のマシンへ移動した後にオリジナルのエージェントが backup メソッドを実行したときである．

### 3.5 バックアップ情報を保存するテーブル

エージェントの移動指示が出されたとき，指定されたエージェントの固有 ID に対応するバックアップファイルを読み込む必要がある．そのために，バックアップが作られるときにエージェントの固有 ID とバックアップファイル名をセットでテーブルに登録し，エージェントの固有 ID からバックアップファイルの名前を参照できるようにする．既にバックアップが存在するエージェントが新しくバックアップを作成した時は最新のバックアップを残して直前のバックアップを削除し，テーブルのファイル名も更新する．

## 4. 移動指示

エージェントの移動指示はシェルから move コマンドによって行う．

move オプション エージェント ID 移動先アドレス

移動指示の対象となるのは最新のバックアップであるが，ユーザはそれを，移動指示が出された時点で最新のものか，移動指示が出された直後に作成されたバックアップなのかを選択できるようにした．この選択はオプションによって行うことができ，「+」をつけた場合は移動指示の直後に作成されるバックアップを，オプションをつけない場合はその時点で最新のバックアップを移動指示の対象となるようにした．また，シェルからの命令による移動だけでなく，この API を通常のアプリケーションに組み込み，move コマンドと同様に，引数としてオプション，エージェント ID，移動先アドレスを渡すことによって，エージェントの移動が可能である．

## 5. 実行結果

バックアップを利用して，指示したエージェントの移動が行われているかを確認するため，図 5 のテストプログラムを作成した．このプログラムは MoveTest という名のエージェントである．12 行目の runAgent メソッドはすべてのエージェントに実装されるエージェントのメインメソッドであり，エージェントの実行とは，この runAgent メソッドが実行されることである．

テストプログラムでは，「1 番目の処理完了」，「2 番目の処理完了」，…と 5 秒おきに出力する度にバックアップを取り，「5 番目の処理完了」まで出力するプログラムである．この実行テストでは，図 6 のように「3 番目の処理完了」と出力されたところで移動指示を出し，確認を行った．図 6 において，「4 番目の処理完了」と出力されているのは，バックアップ移動後もオリジナルのエージェントが自分のマシンで動作し続けているためである．オリジナルのエージェントは，48 行目でフラグが false となり，49 行目に実行する backup メソッドにおいてバックアップを作成しない．そして 50 行目で throw され，処理を停止する．

図 7 の受信側マシンの実行結果を見ると，「4 番目の処理完了」からエージェントの実行が再開されていることがわかる．送信側マシンで「3 番目の処理完了」と出力されたところでバックアップを取り，それを移動させているため，エージェントが移動後に正しい位置から再開できていることが確認できた．

## 6. おわりに

今回作成したエージェントの移動制御を行うための API によって，エージェントがシェルからの指示によっていつでもマシン間を移動することができるようになった．これにより，今後の AgentSphere の開発において，動作テストの効率が上がると考えられる．また，この API を様々なアプリケーションに組み込むことで，実際にユーザが AgentSphere を運用する際の負荷調整等にも利用できるようになった．

```
1.  public class MoveTest extends AbstractAgent{
2.      private static final long serialVersionUID = 1L;
3.      boolean[] FLAG=new boolean[9];
4.      boolean conflag=true;
5.      int BACKUPNUM=0;
6.      int num=1;
7.      int x=10;
8.      int i=0;
9.      MoveTest(){
10.         for(int i=0;i<FLAG.length;i++)
11.             AGENTBACKUP_FLAG[i]=false;
12.     }
13.     public void runAgent() {
14.       try{
15.         if(!FLAG[3]){
16.             if(!FLAG[2]){
17.                 if(!FLAG[1]){
18.                     if(!FLAG[0]){
19.                         print("1番目の処理完了");
20.                         FLAG[0]=true;
21.                         conflag=BackupStatus.continueFlag();
22.                         if(AgentAPI.backup(this,++BACKUPNUM,conflag))
23.                             throw new Exception();
24.                     }else{
25.                         //ローカル変数復元部
26.                         FLAG[0]=false;
27.                     }
28.                     print("2番目の処理完了");
29.                     FLAG[1]=true;
30.                     conflag=BackupStatus.continueFlag();
31.                     if(AgentAPI.backup(this,++BACKUPNUM,conflag))
32.                         throw new Exception();
33.                 }else{
34.                     //ローカル変数復元部
35.                     FLAG[1]=false;
36.                 }
37.                 print("3番目の処理完了");
38.                 FLAG[2]=true;
39.                 conflag=BackupStatus.continueFlag();
40.                 if(AgentAPI.backup(this,++BACKUPNUM,conflag))
41.                     throw new Exception();
42.             }else{
43.                 //ローカル変数復元部
44.                 FLAG[2]=false;
45.             }
46.             print("4番目の処理完了");
47.             FLAG[3]=true;
48.             conflag=BackupStatus.continueFlag();
49.             if(AgentAPI.backup(this,++BACKUPNUM,conflag))
50.                 throw new Exception();
51.         }else{
52.             //ローカル変数復元部
53.             FLAG[3]=false;
54.         }
55.         print("5番目の処理完了");
56.         conflag=BackupStatus.continueFlag();
57.         System.out.println("終了しました");
58.         if(AgentAPI.backup(this,++BACKUPNUM,conflag))
59.             throw new Exception();
60.     }catch(Exception e){
61.         System.out.println("移動しました");
62.     }
63.   }
64.   private void print(String string) {
65.     try {
66.         wait(5000);
67.     } catch (InterruptedException e) {
68.         e.printStackTrace();
69.     }
70.     System.out.println(string);
71.   }
72. }
```

図 5.テストプログラム

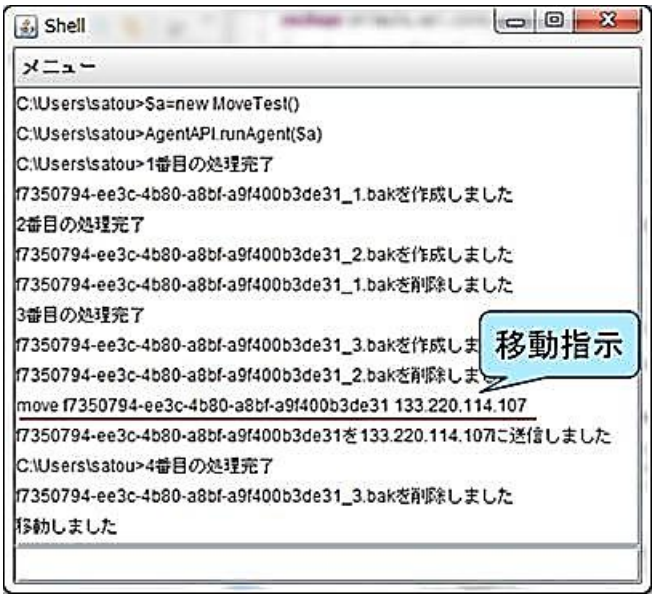


図 6.送信側マシンの実行結果

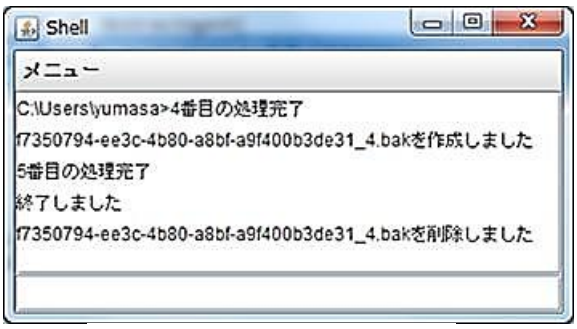


図 7.受信側マシンの実行結果

## 参考文献

[1] 大久保秀・甲斐宗徳「モバイルエージェントシステム AgentSphere の開発―デバッグ補助機能を持つオブジェクト操作可能なシェルの開発―」,FIT2012, B-011,Sep.2012.
[2] 加藤史彬・近藤敬宏・甲斐宗徳「強マイグレーションモバイルエージェントの自己バックアップ機能とエージェント間通信の実装」,FIT2009, B-010, Sep.2009.
[3] 加藤史彬・田久保雅俊・櫻井康樹・甲斐宗徳「コード変換による強マイグレーション化モバイルエージェントの実現」,FIT2007, B-024, Sep.2007.