移動エージェント記述ライブラリ MOBA

首藤 一幸

1997 年 10 月 8 日

Abstract

私が作成している、Java 用移動エージェント記述ライブラリ MOBA(Mobile Agents Framework for Java Language Environment) について述べる。顕著な特徴は、一文で記述できる移動エージェント自身による移動、移動エージェント自身に何の記述も要らない他スレッドからの移動を実現していることである。これにより、移動エージェントの記述が非常に容易になった。

1 はじめに

Java 仮想マシン (JavaVM) 上で実行されるバイトコードは機種独立である。その可搬性に着目して、移動エージェントを記述する枠組 (以下、単に枠組) を Java 言語環境上に構成しようという動きが盛んである。

エージェントの定義は様々で文脈に依存する。移動エージェントと言った場合、コード+データであるオブジェクトだけではなく、実行コンテキストまで含めるべきだと私は考える。

既存の枠組 [1][2][4] において実現されている移動とは実は、実行コンテキストの移動ではなく、コード, オブジェクト (データ) の移動+遠隔メソッド呼び出しである。そのため、移動するだけのために、ローカルに動かすプログラムの作成とは異なった作法が必要となる。エージェントの記述者に各ライブラリ独特の作法を要求する。

本ライブラリ、MOBA[3] では移動エージェントはスレッドである。エージェントの移動はスレッドの移動で実現している。エージェントの記述者は、通常スレッドを作成するように Runnable インタフェースを実装したクラスを記述すればよい。その際、ライブラリ独自の作法が要求されない。エージェント (スレッド) が自身で移動する際は一文で済み、他者 (スレッド) に移動させられる場合は、移動させられることを考慮した記述は一切不要である。

本稿では、他の枠組と比較しての MOBA の特徴、要素技術、問題を主に述べる。

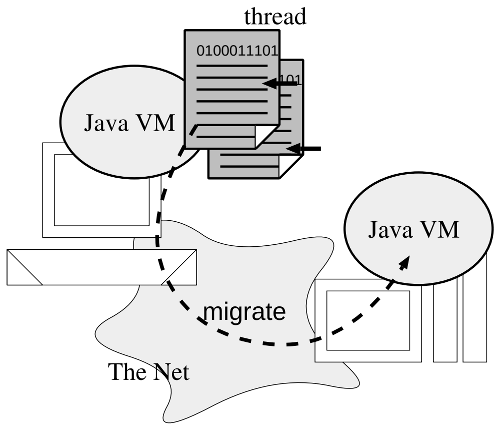

図 1: Thread Migration

2 既存の各種枠組の特徴

MOBA の比較対象となる各社の枠組を概観する。

MOBA

私が作成している、移動エージェント記述ライブラリ。

ネットワークを越えたスレッドの移動が可能な唯一のソフトウェア。スレッドの移動という形で、移動エージェントの容易な記述を実現。

Kafka[2]

富士通研究所の西ヶ谷らが作成している、マルチ・エージェント記述ライブラリ。

処理をメソッド単位ではなく、クラス Action のサブクラス単位で扱うことで、エージェントに対する柔軟なリフレクション、処理に対するアクセスコントロールを実現している。分散ディレクトリを用意している。

Aglets Workbench[1]

IBM で作成されている、移動エージェントのプログラミング環境。

ホスト間を移動可能なオブジェクト aglet を定義している。利用しやすいように、旅程を指定しての移動など典型的な利用パターンをあらかじめクラスとして用意している。

Voyager[4]

分散システムの開発基盤として ObjectSpace が開発しているライブラリ。

エージェントの遠隔生成が可能であったり、同一であるべきオブジェクトが複数ホスト上で参照される場合を考慮していたり、エージェントの移動だけではない広範な応用を指向している。

3　移動エージェントの記述法

移動エージェントの記述法を、Voyager と MOBA で比較する。

Voyager

`Agent` のサブクラスが自分で、移動先で実行したいメソッドを指定して `moveTo()` メソッドを呼ぶ。

```
import COM.objectspace.voyager.*;
public class MobileAgent extends Agent {
  public void message(Agent myOwn, String s) {
    System.out.println(s);
  }

  public void go(String address) {
    moveTo(address, "message", "Hello !");
  }
}
```

この様に、オブジェクトの移動+メソッドの遠隔呼び出しとなっている。

MOBA

移動エージェントは、通常のスレッド作成と同様にまずインタフェース Runnable を実装したクラスを記述する。次に、通常の `java.lang.Thread` クラスの代わりに `MobaThread` クラスでスレッドを生成する。MobaThread で生成したスレッドは移動が可能である。

自分自身の移動　自分で `MobaThread.goTo(移動先)` を呼ぶ。移動先では `goTo()` の次の文から実行が開始される。

```
...
MobaThread.goTo(移動先ホスト);
...
```

他スレッドの移動　移動させたいスレッドに対して `moveTo(移動先)` を呼ぶ。移動させられたスレッドは移動後、移動直前の続きを実行する。

```
MobaThread t = new MobaThread(...);
t.start();
...
t.moveTo(移動先ホスト);
```

エージェントの移動として、MOBA 以外ではデータが移動しメソッドの遠隔呼び出しが行われるのに対して、MOBA では実行コンテキストも移動する。ゆえに MOBA では移動エージェントの記述に特別な作法を必要としない。自身の移動には `goTo(移動先)` の呼び出し、他者 (他スレッド) の移動には `moveTo()` の呼び出しだけで済む。

4　動作環境

UNIX 互換 OS 上で Java Development Kit(JDK) が必要。JDK のバージョン 1.1.×、1.0.2 双方に対応。

JDK	1.1.× or 1.0.2
OS	Linux, SunOS 5, IRIX,...
必要なツール	GNU make

5　要素技術

MOBA の実現に、次の技術要素を用意した。紙面の都合上、各技術の詳細は割愛する。

Reflection

クラスやオブジェクト自身に関する情報を扱う技術。

Object Marshaling

オブジェクト (のグラフ) と対応するバイト列の相互変換を行う技術。

Reflection, Object Marshaling のライブラリは現在 JDK 1.1.×にも含まれている。

Thread Externalization

スレッドが保持する実行コンテキスト、スレッドから参照され得るオブジェクト群と対応するバイト列の相互変換を行う技術。

Class Exportation

JavaVM 内にあるクラス定義情報からクラスを定義するバイト列 (クラスファイル) を salvage, 復元する技術。

	転送プロトコル	移動エージェント	製作者
MOBA	独自	MobaThread,(Runnable)	首藤
Kafka	RMI or HORB	Action,(Agent)	富士通研究所
Aglet	ATP[1]	Aglet,...	IBM
Voyager	独自	Agent	ObjectSpace

図 2: 各枠組の比較

OS のファイルシステムからクラスファイルを検索すればよいので、Thread Migration に必ずしも必要なわけではない。Class Exportation によって、一旦 JavaVM 内にロードされたクラスをファイルシステムから再び検索するといった無駄を省ける。

Thread Migration

スレッドを JavaVM 間、マシン間で移動させる技術。

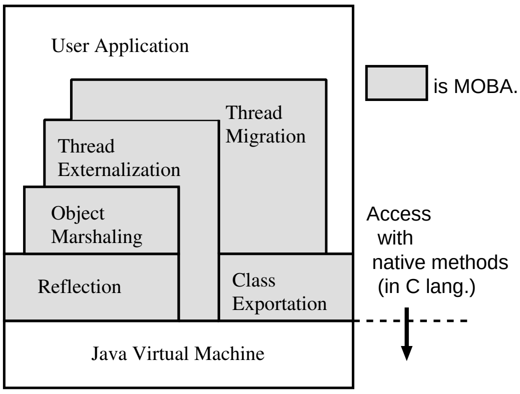

図 3: MOBA の要素技術

6 応用

移動先マシンで処理を行う移動エージェントの記述以外にも、以下の様な応用が考えられる。

いずれも、他スレッドの移動 (`moveTo()` メソッド) を利用すれば、目的の処理を行うコードの中で移動について考慮する必要は一切ない。他の枠組ではそうはいかない。

load balancing

負荷の軽い、あるいは処理の速いマシンへ移動する、させる。

耐障害性の向上

電源断などマシンの異常を感知して、自身のファイルへの保存や、他のマシンへの移動を行う。

旅に出るスレッド

育って帰って来るか、くたばるか。

7 問題

7.1 将来解決予定の問題

セキュリティ

スレッドの受け入れ側に受け入れスレッドの認証機能が必要。何を key にどのように認証するか。

受け入れたスレッドができることを制限する必要がある。SecurityManager によってアプレットに対して制限をかけている手法を参考にするか。

オブジェクトの同一性

現在、オブジェクトの移動はすべてコピーで実現している。オブジェクト A,B がオブジェクト C を参照しているとすると、B が移動すると C もコピーされて移動していく。B が元の JavaVM に戻って来たとき、A が参照する C と B が参照する C は別物となっている。

Just In Time(JIT) コンパイラへの対応

JavaVM が JIT コンパイラを持つ場合、メソッドはプロセッサネイティブなコードに変換される可能性がある。現状、ネイティブなコードを実行中のスレッドを考慮していない。

7.2 解決が困難な問題

MOBA の実装が **JDK** に大きく依存している

JDK 付属 JavaVM の内部構造に大きく依存している。JDK 付属のもの以外の JavaVM で使えるかは疑問、未確認である。

native methods を使用

C で記述されているメソッドがあるので、アプレットから利用するためにもローカルディスクへのインストールが必要。

スタック上要素の型の判別

実行コンテキストをバイト列へ変換するためには、スタック上の要素 (32bit) の型 (整数, オブジェクト...) を判別する必要がある。しかし、JavaVM 内にスタック上要素の型情報がない。

理論的には、メソッドのコードを頭から追うことである時点のスタックの状態は割り出

せる。しかしこの手法は実行時間の点から現実的ではない。

現在は、スタック上の値をある型だと仮定し、シグナル segmentation voilation をとらえることで仮定が正しかったかどうかを判定している。この手法では判別結果が常に正しいとは限らない。

JavaVM の互換性

JDK の各バージョン間で、JavaVM の互換性が完全ではない。そのため、クラス定義を流用できない場合がある。

例えば、JDK1.1 からはメソッド呼び出しの性能を稼ぐため実行中に JavaVM が `invoke*` → `invoke*_quick` とメモリ中のコードを書き換える。JDK1.0.2 の JavaVM には `invoke*_quick` 命令はない。JDK1.1 から JDK1.0.2 の JavaVM にクラス定義情報を転送しても JDK1.0.2 では実行できないことがあり得る。

7.3 些細な問題

スレッドと共に定義情報の転送が必要なクラスの羅列法

スレッドの移動において、スレッドの情報を移動する前に、移動元には存在して移動先に存在しないクラスの定義情報を転送、移動先でクラスを定義している。定義情報の移動が必要なクラスとして、転送元スレッドが参照し得るクラス情報を羅列する必要がある。

スレッドが参照し得るクラス群は移動する時点に動的に求める必要がある。しかし現在は、クラス情報から静的に得ている。このため、例えばクラス `Bar` がクラス `Foo` を継承しているとして

```
Foo var = new Bar();
```

というコードがあった場合に、移動先でクラス `Bar` が必要となることを見抜けない。

同名で定義が異なるクラスの扱い

移動元が異なる 2 つのスレッドは、同じ名前で定義の異なるクラスを要求する可能性がある。

現在は、スレッド受け入れデーモンは ClassLoader をひとつしか持たないので対応できない。将来は ClassLoader を適切に複数持つことで解決する。

Class Exportation

JavaVM 内からクラス定義情報を拾い出す限り、Exporter(クラス ClassWriter) が対応していない attribute が無視されてしまう。(例: クラスファイルに埋め込まれたプロセッサネイティブなコード)

クラスファイル内には存在するが、JavaVM 内には存在しない情報がいくつかある。幸い、そのどの情報も実行には影響しないものなので問題は起きていない。

8 今後

上記の諸問題を解決していくと共に以下を行う。

パフォーマンスの評価

実行時間を評価する。特に load balancing に利用するためには性能が要求される。

スレッド受け入れデーモン (**ThreadField**) の改良

次の機能を追加する。

- 接続元の認証
- 受け入れたスレッドの実行時間を制限する機能

スレッドフィールドの構成、利用技術の開発

次を開発する。

- ThreadField をネットワーク上から探すソナー
  ネットワーク上に broadcast することで ThreadField を探す。
- ソナーの音波増幅
  broadcast ドメインの外の ThreadField の代理でソナーに反応する。

参考文献

[1] IBM Aglets Workbench. http://www.trl.ibm.co.jp/aglets/index-j.html.

[2] Kafka: Java 用マルチ・エージェント記述ライブラリ. http://www.fujitsu.co.jp/hypertext/free/kafka/jp/.

[3] MOBA – Mobile agent framework for Java language environment. http://www.shudo.net/software/index-j.html.

[4] ObjectSpace: Voyager Overview. http://www.objectspace.com/Voyager/.