モーバイルエージェント記述言語 APSL

勝野 恭治[†]　　村田 賢一[††]　　所 真理雄[†††]

katsuno@mt.cs.keio.ac.jp　　*murata@csl.sony.co.jp*　　*mario@mt.cs.keio.ac.jp*

† 慶應義塾大学大学院 理工学研究科
†† ソニーコンピュータサイエンス研究所
††† ソニー株式会社 / 慶應義塾大学

Abstract

現在，Java を用いてモーバイルエージェントを実装するのが最適であると考えられており，Java をベースとしたモーバイルエージェント記述言語が数多く実装されている．しかし，既存の Java をベースとしたモーバイルエージェント記述言語にはホスト間の移動，エージェント間の通信，変数のアクセス制御といった記述がそれぞれ複雑になるという問題点がある．そのため，プログラマはモーバイルエージェントを容易に記述することができない．本論文では，これらの問題点を解決する Java をベースとしたモーバイルエージェント記述言語 `APSL` (Agent Programming System and Language) を提案する．プログラマは `APSL` を用いるとモーバイルエージェントを容易に記述することができる．

1　はじめに

近年，無線ネットワークを用いてモーバイルコンピュータをインターネットに接続して，インターネット上で作業を行う機会が増加している．無線ネットワークは有線ネットワークに比べ信頼性が低く，通信速度が遅い．そのため，無線ネットワークを使用する時間，即ちモーバイルコンピュータとインターネットとの接続時間を短縮したいという要望がある．現在その要求を満たす手法として，ネットワーク上を動き回るモーバイルエージェントが注目されている．ユーザは，モーバイルエージェントを用いるとモーバイルエージェントをインターネットに送り出した後モーバイルエージェントが戻ってくるまでの間，自分がいるホストとインターネットとの接続を切ることができる．そのため，モーバイルコンピュータとインターネットとの接続時間を短縮することができる．

一般にモーバイルエージェントは，ホスト間を移動し移動先のホスト上の様々なアーキテクチャ，OS の上で実行する．これに対して Java[3] は，オブジェクトを他のホストに移動させ移動先のホスト上で実行させることができる．また，Java のプログラムはアーキテクチャ，OS に依存しない．そのため，現在 Java はモーバイルエージェントの実現に適していると考えられている．現在，Aglet[1]，Java-To-Go[6] など Java をベースにしたモーバイルエージェント記述言語が数多く実装されている [7]．これらの言語は全て Java のライブラリとして実装されている．そのため，プログラマは新しい構文を覚える必要がない．しかし，次の 3 点の記述が複雑になるため，プログラマはモーバイルエージェントを容易に記述することができない．

- ホスト間の移動

  Java のオブジェクトはローカル変数の値やメソッドがどこまで進んだかを示す実行カウンタの値を保持したまま他のホストに移動することができない．そのためプログラマは，モーバイルエージェントのホスト間の移動を記述する際に，ローカル変数や実行カウンタの値の保持も記述しなければならない．

- エージェント間の通信の記述

  エージェント間の通信は一般に図 1のような状態遷移図で表わされる．そして，通信を行なうエージェントは予め状態遷移図，即ち通信プロトコルを決めておかないと通信を行なうこと

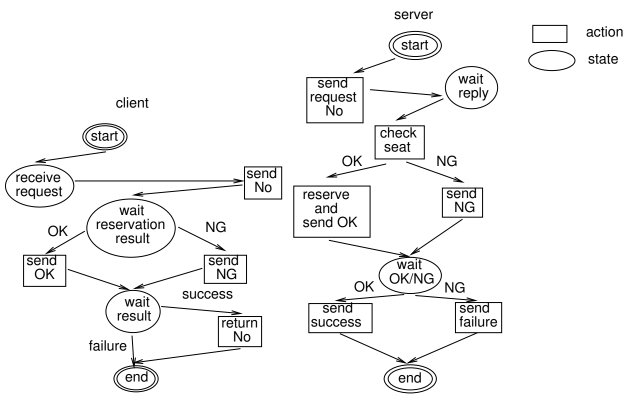

図 1: エージェント間の通信の例

ができない．そのため，プログラマは通信で用いる通信プロトコルを事前に知らなければならない．また，送られてきたメッセージの内容をエージェントが保持する変数に格納する際にキャストしなければならないので，プログラマは各通信で用いられるメッセージフォーマットも知らなければならない．しかしメッセージフォーマット，通信プロトコルはサービスを提供するホストやモーバイルエージェントが通信で用いるリソースの状況によって異なる．そのため，プログラマがこれらを全て事前に知るのは大変困難である．

- 変数のアクセス制御

  エージェントはソケット通信によって他のエージェントが持つ変数にアクセスする．そのため，変数のアクセス制御がたとえ `private` でもソケットを通じてアクセスされてしまう危険がある．よって，ソケット通信による変数へのアクセスを制御する必要がある．しかしこれを Java のライブラリで実装すると，アクセス制御された変数とアクセス制御されていない変数とをプログラム中で透過に扱うことができなくなってしまう．

本論文では，プログラマのモーバイルエージェントを記述する際の負担を減らすために，Java をベースとしたモーバイルエージェント記述言語 `APSL`(Agent Programming System and Langauge) を提案する．`APSL` では次の 3 点を行なうことによってプログラマの負担を減らしている．

- Java の構文の拡張

  `APSL` では，ホスト間の移動，エージェント間の通信，変数のアクセス制御を容易に記述できるように Java の構文を拡張している．これによってプログラマはモーバイルエージェントを容易に記述することができる．

- Java フロントエンドコンパイラの提供

  Java フロントエンドコンパイラは，`APSL` で拡張した構文から Java のコードを生成する．そのため，既存の豊富な Java のライブラリを `APSL` で用いることができる．

- 通信プロトコルを動的に選択する機構の導入

  `APSL` では，獲得したいサービスと通信で用いるリソースを指定するだけで，状況に最も適した通信プロトコルを動的に選択し用いることができる．また，各通信で用いられるメッセージフォーマットは選択した通信プロトコルの中に記述されている．そのため，プログラマは通信で用いる通信プロトコルと各通信で使用されるメッセージフォーマットを意識する必要がない．

2 関連研究

既存のモーバイルエージェント記述言語は，Java をベースとした言語と Java をベースとしていない言語の 2 つに大きく分けられる．本章では，Java をベースとした言語の中から代表的な言語として Aglet[1], Java-To-Go[6] を，Java をベースとしていない言語の中から代表的な言語として Telescript[9][10]，Agent Tcl[4][5] をそれぞれ紹介する．

Aglet は日本アイビーエム東京基礎研究所で開発された言語で，エージェント間の非同期なメッセージ通信を支援している．Aglet は Java RMI ライブラリ [2] を使用してモーバイルエージェントの移動を実装しており，ローカル変数や実行カウンタの値を保持したままモーバイルエージェントを他のホストに移動させることができない．そのためプログラマは図 2のように `switch` 文や `if` 文と実行カウンタを表す変数を用いて実行カウンタの値を保持したり，ローカル変数をインスタンス変数に変換してローカル変数の値を保持したりする必要がある．ま

```
try {
    switch ( _state ) {

    case HOST1:
        .....
        _state = HOST2;
        dispatch( host2 );
        break;

    case HOST2:
        .....
        break;

    default:
        throw new AgletException("Error");
    }
} catch (AgletException ae) {
    .....
}
```

図 2: Aglet の移動の記述

```
AgletProxy proxy;
.....
public void func( ){
    Message msg;
    .....
    proxy.sendMessage( msg );
    .....
}
....
public boolean handleMessage( Message msg ){
     .....
}
```

図 3: Aglet の通信の記述

た Aglet では，プログラマはメッセージを受け取る処理を `handleMessage` メソッド内に記述しなければならない (図 3)．そのため，通信プロトコルは固定している．また，メッセージを送受信するエージェント間でメッセージフォーマットを予め決めておく必要がある．

```
try {
    if (nextHost == null)
        local.go(this, result);
    else
        local.go(nextHost, this, result);
    } catch (IOException e) {
        local.println("Problem: ");
        local.println( e.getMessage( ) );
    }
}
```

図 4: Java-To-Go の移動の記述

```
public void func1( ){
    Message msg1;
    .....
    CommHandle ch = getCommHandle("getmail");
    ch.write( msg1 );
    .....
    Message msg2 = ch.read( );
    String str = ( String )msg2.content( );
    .....
}
```

図 5: Java-To-Go の通信の記述

Java-To-Go はカリフォルニア大学バークレー校で開発された言語で，マルチキャスト通信を行うことができる．Java-To-Go は独自の Java RMI ライブラリを用いてモーバイルエージェントの移動を実装しているが，Aglet と同様にモーバイルエージェントの移動時にローカル変数や実行カウンタの値を保持することができない．そのため，プログラマは Aglet と同様な記述を行なわなければならない (図 4)．また Java-To-Go では Aglet と同様にメッセージを送受信するエージェント間でメッセージフォーマットを予め決めておかなければならないが，Aglet とは異なりメッセージを受け取るメソッドは特に決められていない (図 5)．そのため，`java2go.util.Message` クラスを拡張することによって通信プロトコルを変えることができる．しかし，言語側で通信プロトコルの変更を支援していないので，大変困難であると考えられる．

Telescript はジェネラルマジック社で開発された言語で，独自のインタプリタ言語として実装されている．Telescript はモーバイルエージェント記述言語の走りとして広く知られている．そして Telescript は，Aglet や Java-To-Go と違いローカル変数や実行カウンタの値を保持したままモーバイルエージェントを他のホストに移動させることができる．

Agent Tcl はダートマス大学で開発された Tcl[8] をベースとした言語で，Tcl の仮想機械部分を一部修正している．Agent Tcl はユーザがいるホストとネットワークとの切断や再接続を行う機構の実現に重点を置いている．そして，Agent Tcl では，Tcl の仮想機械を一部修正して内部状態を保持させたままスクリプトを他のホストに移動させている．そのため，Agent Tcl は Telescript と同様にローカル変数や実行カウンタの値を保持したままモーバイルエージェントを他のホストに移動させることができる．

3 設計

3.1 APSL のシステムモデル

`APSL` はモーバイルエージェント，プレースエージェント，コミュニケーションズパッケージという

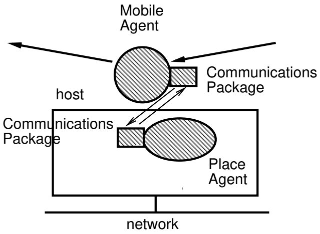

図 6: APSL の構成要素

3 つの要素から構成されている (図 6) .

- モーバイルエージェント

  ネットワーク上のホスト間を渡り歩き，サービスを獲得するためにホスト上のプレースエージェントと通信を行うプログラム．

- プレースエージェント

  モーバイルエージェントに様々なサービスを提供するプログラム．プレースエージェントは各ホスト上に 1 つづつ存在し，他のホストに移動することはない．

- コミュニケーションズパッケージ

  メッセージの送受信や状態の遷移が記述されたエージェント間の通信を支援するプログラム．コミュニケーションズパッケージにはモーバイルエージェント用とプレースエージェント用があり組になっている．各エージェントがコミュニケーションズパッケージを実行すると，コミュニケーションズパッケージ間で自動的に通信を行ない，通信の結果が各エージェントに返る．また，コミュニケーションズパッケージはエージェントが持つ変数にアクセスしたりメソッドを呼び出したりすることができる．

続いて，APSL が用いられるネットワーク環境を図 7に示す．APSL が用いられるネットワーク環境には，プログラマによって作成されたモーバイルエージェントを蓄えているエージェンシーホスト，モーバイルエージェントに対して様々なサービスを提供するサービスホスト，モーバイルエージェントを用いるユーザがいるカスタマーホストという 3 つのホストがそれぞれ存在する．

最後に，図 7のネットワーク環境下で用いられるシナリオを説明する．

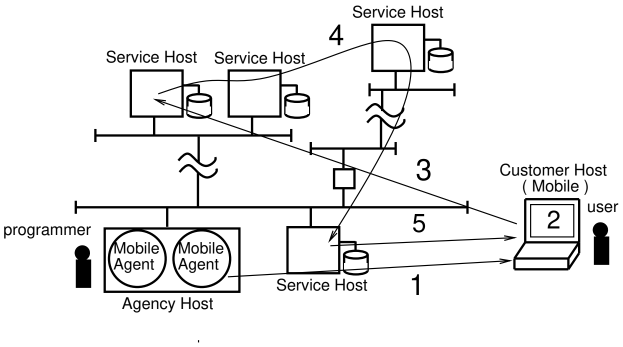

図 7: APSL が用いられるネットワーク環境

1. ユーザはカスタマーホストをネットワークに接続する．そして，エージェンシーホストからモーバイルエージェントをダウンロードする．
2. ユーザはモーバイルエージェントにパラメータと全ての作業が終わった際に移動するサービスホストのホスト名を与える．
3. ユーザはモーバイルエージェントをネットワークに送り出す．送り出した後，ユーザはカスタマーホストをネットワークから切断する．
4. モーバイルエージェントはサービスホストを渡り歩き，サービスホストから様々なサービスを獲得する．そして，全ての作業を終えるとユーザに指定されたサービスホストに移動する．
5. ユーザはカスタマーホストを再びネットワークに接続し，モーバイルエージェントが戻って来ているかどうかを 2. で指定したサービスホストに問い合わせる．もし戻って来ていればモーバイルエージェントをカスタマーホストに移動させ結果を表示させる．もし戻って来ていなければカスタマーホストをネットワークから切断し，モーバイルエージェントが戻って来るまで 5. の作業を繰り返す．

3.2　ホスト間の移動

APSL では，モーバイルエージェントが他のホストへ移動する際にローカル変数と実行カウンタの値を自動的に保存している．そのため，モーバイルエージェントは移動先のホストで実行を再開することができる．

3.3 エージェント間の通信

現時点のAPSLでは，エージェント間の通信はモーバイルエージェントとプレースエージェントの2者間のみで，その際に必ずコミュニケーションズパッケージを使用することを想定している．また現時点のAPSLでは，モーバイルエージェント用，プレースエージェント用のコミュニケーションズパッケージを持っているのはプレースエージェントのみである．そのため，エージェント間で通信を行うためにはプレースエージェントがモーバイルエージェントにモーバイルエージェント用のコミュニケーションズパッケージを渡さなければならない．プレースエージェントが通信で用いるコミュニケーションズパッケージを選択しモーバイルエージェントに渡すプロトコルを以下に示す．

1. モーバイルエージェントはプレースエージェントに獲得したいサービスの名前を送る．
2. プレースエージェントはそのサービスを提供する際に必要なリソースのリストをモーバイルエージェントに送る．
3. モーバイルエージェントは送られて来たリスト内のリソースを通信で用いることができるかどうか調べる．そして，その結果をプレースエージェントに送る．
4. プレースエージェントは，モーバイルエージェントから送られてきた結果を元に最も適したコミュニケーションズパッケージを選択しモーバイルエージェントに送る．もし適切なコミュニケーションズパッケージがなければエラーメッセージを送る．

このプロトコルによってプログラマは，使用するコミュニケーションパッケージを意識することなくモーバイルエージェントを記述することができる．

続いて，コミュニケーションズパッケージがエージェントが持つ変数へのアクセスやメソッドの呼び出しを行なうプロトコルを以下に示す．

1. コミュニケーションズパッケージはアクセスする変数の名前，もしくは呼び出すメソッドの名前と引数をエージェントに送る．
2. エージェントは変数，もしくはメソッドのアクセス状態を調べる．
3. もしアクセスを許可していればエージェントは，変数へのアクセスもしくはメソッドの呼び出しを行ない，その結果をコミュニケーションズパッケージに送る．もしアクセスを許可していなければ，エラーメッセージをコミュニケーションズパッケージに送る．

3.4 変数のアクセス制御

各エージェントはそれぞれ変数のアクセス状態を管理するテーブルを持っている．そして，変数へのアクセスが他のエージェントから要求されたら，エージェントは該当する変数のアクセス状態をそのテーブルを用いて調べる．もしアクセスが許可されていればアクセスを行い結果を返す．もし，アクセスが許可されていなければエラーメッセージを返す．APSLで提供するアクセス状態には，read，writeアクセスを共に許可しない，readアクセスのみを許可する，writeアクセスのみを許可する，read,writeアクセスを共に許可する，という4つの状態がある．

また，プログラマは通信の記述の中で変数のアクセス状態を一時的に変更することができる．一時的に変更されたアクセス状態は通信が終了すると以前のアクセス状態に戻る．

3.5 記述例

APSLの記述例として，飛行機の往復のチケットとホテルの予約を予約するのであれば全て予約し予約しないのであれば全く予約しないといった作業を行うモーバイルエージェント，飛行機のチケットの予約とキャンセルを受けつけるプレースエージェント，図1に示されている飛行機のチケットを予約する通信プロトコルを表すコミュニケーションズパッケージの記述例をそれぞれ図12,13,14に示す．

図12から，通信の結果を格納する変数が，予約する時はwriteアクセスのみを許可しており(16, 52行目)，キャンセルする時にはreadアクセスのみを許可しており(22, 29, 35, 59行目)，それ以外はread,writeアクセスを共に許可していない(4, 8行目)ということが分かる．

続いて図13から，予約というサービスを提供する際には3つのコミュニケーションズパッケージの

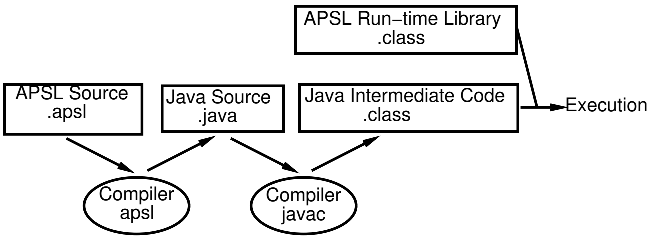

図 8: APSL の実行の手順

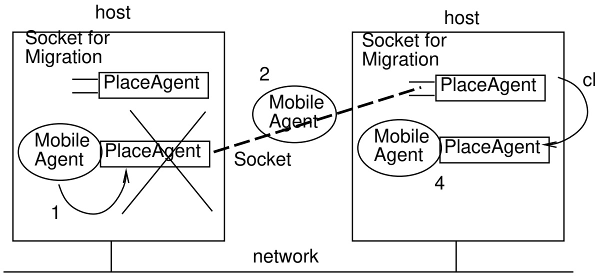

図 9: ホスト間の移動の機構

中から，キャンセルというサービスを提供する際には2つのコミュニケーションズパッケージの中から，モーバイルエージェントが通信で用いることができるリソースの状況に応じてそれぞれ選択しているということが分かる (11–28 行目)．そして，`check`, `reserve`, `cancel` というメソッドの呼び出しをコミュニケーションズパッケージに対して許可しているということも分かる．

また，図 12, 13から，図 12のモーバイルエージェントが図 13のプレースエージェントと通信を行う際には，`RESERVATION` サービスの場合は `Normal`，`CANCEL` サービスの場合は `Cancel` というコミュニケーションズパッケージがそれぞれ用いられるということが分かる．

最後に図 14から，このコミュニケーションズパッケージはプレースエージェントが持つ `check`, `reserve` というメソッドをそれぞれ呼び出すということが分かる (70, 81 行目)．

4 実装

本論文では，プログラムをリンクする際に用いられる APSL ランタイムライブラリ[1]と APSL のソースから Java のソースを生成する Java のフロントエンドコンパイラ，APSL コンパイラ[2]をそれぞれ実装して APSL を実現した．APSL のソースプログラムが実行されるまでの手順と図 12のプログラムのコンパイルされた結果をそれぞれ図 8，図 15に示す．

4.1 ホスト間の移動

APSL では，Java RMI ライブラリを用いてモーバイルエージェントのホスト間の移動を実装した．しかし，Java RMI ライブラリには以下の 2 つの問題があり，モーバイルエージェントが移動先のホストで実行を再開することができない．

(a) オブジェクトのローカル変数の値を保持できない．

(b) オブジェクトの実行カウンタの値を保持できない．

これらの問題が生じるのは Java RMI ライブラリがオブジェクトのインスタンス変数内のデータのみをシリアライズしているためである．そのため，APSL ではこれらの問題をそれぞれ以下のように解決した．

(a) APSL コンパイラがモーバイルエージェントの `main` メソッド内の全てのローカル変数をインスタンス変数に変換する．

(b) APSL コンパイラが `main` メソッドがどこまで実行されたかを示す変数と `switch` 文を生成する．

続いて，ホスト間の移動の機構を図 9を用いて説明する．

0. 各ホスト上には well-known ポート[3]で公開しているソケットを持つプレースエージェントが 1 つづつ存在している．そして，モーバイルエージェントはそれぞれ `clone` メソッドで生成されたプレースエージェントの上にいる．

---

[1] JDK 1.1.4 を使用

[2] bison 1.25, flex 2.5.4, gcc 2.7.2.1 を使用

[3] ポート番号 7777

1. モーバイルエージェントは，移動先のホストのホスト名と自分自身を示すポインタを今いるプレースエージェントに送る．

2. ホスト名とポインタを受け取ったプレースエージェントは移動先のホスト上の well-known ポートで公開しているソケットを持つプレースエージェントにソケットの作成を要求する．ソケットが作られると，移動先のプレースエージェントは自分の clone を作成するために `clone` メソッドを実行する．

3. 移動先の clone のプレースエージェントはソケットを通って移動して来たモーバイルエージェントを受け取る．移動元のモーバイルエージェントがいたプレースエージェントは，モーバイルエージェントの移動が終わり次第消滅する．

4. 移動先の clone のプレースエージェントは移動して来たモーバイルエージェントのmainメソッドを実行させる．すると，モーバイルエージェントは実行を再開する．

4.2　エージェント間の通信

コミュニケーションズパッケージは，コミュニケーションズパッケージ同士で通信を行なっている最中に，エージェントが持つ変数にアクセスしたりメソッドを呼び出したりする場合がある (図 1, 14)．これを実現するためには，モーバイルエージェント，プレースエージェント，コミュニケーションズパッケージがそれぞれスレッドとして実行していなければならない．そのため `APSL` では，モーバイルエージェント，プレースエージェント，コミュニケーションズパッケージそれぞれをスレッドとして実行させ，通信を行なうためのソケットを持つように実装した (図 10)．

4.3　変数のアクセス制御

`APSL` では`_apsl_AccessTable` クラスと`_apsl_Variable` クラスを用いて，変数のアクセス制御を実装した．`_apsl_Variable` クラスはインスタンス変数の名前，値，アクセス状態を保持している．そして，`_apsl_AccessTable` クラスは`_apsl_AccessTable` クラスで表されている各インスタンス変数をハッシュテーブルとして管理している．`_apsl_Variable` クラスのプログラムを図 11 に示す．

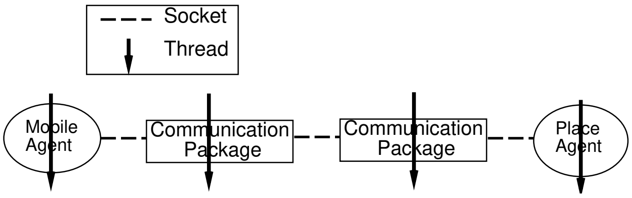

図 10: エージェント間の通信

```
import java.io.*;
import java.util.*;

public class _apsl_Variable extends Object
        implements Serializable, Cloneable {

    private String  name;
    private Object  value;
    private int     access;

    Variable( ..... ){
        .....
    }
    .....
}
```

図 11: Varible クラス

`name` は変数の名前，`value` は変数に格納されている値，`access` は変数のアクセス状態をそれぞれ示している．`access` フィールドの値とその値に対するアクセス状態を次に示す．

0	read, write アクセスを共に許可しない
1	read アクセスを許可する
2	write アクセスを許可する
3	read, write アクセスを共に許可する

5　今後の課題

現時点での `APSL` では，モーバイルエージェントが他のホスト上に存在するプレースエージェントと通信を行なうのに，そのプレースエージェントがいるホストに移動して通信を行わなければならない．しかし現実には，移動して通信を行うよりも今いるホストから通信を行う方が通信コストがかからない場合がある．そのため，通信コストを計算し，移動して通信を行うか，それとも今いるホストから通信を行うかを動的に選択する機構が必要である．これ

を実現するためには，APSL ランタイムライブラリが通信コストを計算する機構を持っていなければならない．

また現時点の APSL では，コミュニケーションズパッケージを持つことができるのはプレースエージェントのみである．しかし，モーバイルエージェントもコミュニケーションズパッケージを持つことができれば，モーバイルエージェントが持つコミュニケーションズパッケージを用いて通信を行うことができる．そのため，モーバイルエージェントを記述するプログラマが望む通信を行うことが可能となる．これを実現するためには，コミュニケーションズパッケージを選択するプロトコルと APSL の記述構文を変更する必要がある．

最後に APSL では，不特定多数のモーバイルエージェントがサービスホストに移動してくる．そのため，サービスホストに害を与えるモーバイルエージェントが移動してくることが考えられる．現時点での APSL ではサービスホスト上に存在するプレースエージェントが持つリソースに対するアクセスは制御しているが，CPU，メモリなどといったサービスホスト上のリソースに対するアクセスは制御してない．そのため，サービスホスト上のリソースに対するアクセスを制御しなければならない．これを実現するためには，APSL ランタイムライブラリがホスト上のリソースへのアクセスを制御する機構を持っていなければならない．

6 結論

現在の Java をベースとしたモーバイルエージェント記述言語にはホスト間の移動，エージェント間の通信，変数のアクセス制御の記述がそれぞれ複雑になるという問題点があり，プログラマはモーバイルエージェントを容易に記述することができなかった．そこで本論文では，モーバイルエージェントを容易に記述することができるモーバイルエージェント記述言語 APSL を提案した．APSL ではホスト間の移動，エージェント間の通信，変数のアクセス制御という 3 点について Java の構文の拡張を行った．また，APSL のソースから Java のソースに変換する Java フロントエンドコンパイラ，APSL コンパイラを提供した．そして，通信プロトコルを動的に選択する機構を導入した．

APSL を用いると，プログラマはローカル変数や実行ポインタの値の保持を意識することなく，モーバイルエージェントのホスト間の移動を記述することができる．そして，通信で用いる通信プロトコルや各通信で用いられるメッセージフォーマットを意識しないでエージェント間の通信を記述することができる．さらに，アクセスを制御された変数とアクセスを制御されていない変数をプログラム中で透過に扱うことができる．

参考文献

[1] Daniel T. Chang and Danny B. Lange. Mobile Agent: A New Paradigm for Distributed Object Computing on the WWW. *OOPSLA '96 Workshop*, 1996.

[2] Troy Downin. *RMI: Developing Distributed Java Applications with Remote Method Invocation. IDG Books*, 1997.

[3] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification. Addison Wesley*, 1996.

[4] Robert Gray, Daniela Rus, and David Kotz. Autonomous and Adaptive Agents that Gather Information. *AAAI '96 Workshop on Intelligent Adaptive Agents*, 1996.

[5] David Kotz, Robert Gray, Saurab Nog, Daniela Rus, Sumit Chawla, and George Cybenk. Agent TCL: Targeting the Needs of Mobile Computer. *IEEE Internet Computing*, 1(4), July/August 1997.

[6] William Li and David G. Messerschmitt. *Java-To-Go: Itineractive Computing Using Java.* Technical Report, *Department of Electrical Engineering and Computer Sciences, University of California at Berkeley*, 1996.

[7] David Peterson. *Developing Intelligent Agents Using Java. McGraw-Hill*, 1997.

[8] Brent Welch. *Practical Programming in Tcl and Tk. Prentice Hall*, 1995.

[9] James E. White. Mobile agents make a network an open platform for third-party developers. *IEEE Computer*, 27(11):89–90, Nov 1994.

[10] James E. White, Bradshaw, and Jeffrey(ed.). *Software Agents. AAAI Press*, 1996.

```
agent A1 {
 String air_host = "airplane.jp";
 String[ ] serial = { "123", "136"};
 String[ ]:[-]: air_ID = new String[2];
 String hotel_host = "hotel.jp";
 String checkin = "19970822";
 String checkout = "19970823";
 String:[-]: hotel_ID;
 main(  ){
  boolean air;
  host( air_host ){
   for ( int i = 0; i < 2; i++ ){
    communicate( RESERVATION ){
     serial[ i ]
    }  reply  {
     :[w]:air_ID[ i ]
    }
   }
   if (air_ID[0]==null && air_ID[1]==null){
    for ( int i = 0; i < 2; i++ ){
     communicate( CANCEL ){
      :[r]:air_ID[ i ]
     } reply { }
     air = false;
    }
   }
   else if ( air_ID[ 0 ] == null ){
    communicate( CANCEL ){
     :[r]:air_ID[ 1 ]
    } reply { }
    air = false;
   }
   else if ( air_ID[ 1 ] == null ){
    communicate( CANCEL ){
     :[r]:air_ID[ 0 ]
    } reply {  }
    air = false;
   }
   else { air = true; }
  }
  if ( air == false ){
   host( returnhost ){
    System.out.println("Failure");
   }
  }
  else {
   host( hotel_host ){
    communicate( RESERVATION ){
     checkin_date,
     checkout_date
    }  reply  {
     :[w]:hotel_ID
    }
   }
   if ( hotel_ID == null ){
    host( air_host ){
     for ( int i = 0; i < 2; i++ ){
      communicate( CANCEL ){
       :[r]:air_ID[ i ]
      }  reply  {  }
     }
    }
    host( returnhost ){
     System.out.println("Failure");
    }
   }
   else {
    host( returnhost ){
     System.out.println("Success");
    }
   }
  }
 }
}
```

図 12: モーバイルエージェントの記述例

```
place P1 {
 DB db = new DB( );
 init( ){
  registerCommunication("normal",Normal);
  registerCommunication("member",Member);
  registerCommunication("vip",   Vip);
  registerCommunication("vcancel",VCancel);
  registerCommunication("cancel",Cancel);
 }
 communication( ){
  service( RESERVATION ){
   case[ serial, vip ]:
    start( "vip" );
   case[ serial, member ]:
    start( "member" );
   case[ serial ]:
    start( "normal" );
   default:
    comm_error( );
  }
  service( CANCEL ){
   case[ air_ID, vip ]:
    start( "vcancel" );
   case[ air_ID ]:
    start( "cancel" );
   default:
    comm_error( );
  }
 }
 incomm Boolean check( Vector vec ){
  return( db.check( vec ) );
 }
 incomm Boolean reserve( Vector vec ){
  return( db.reserve( vec ) );
 }
 incomm Boolean cancel( Vector vec ){
  return( db.cancel( vec ) );
 }
}
```

図 13: プレースエージェントの記述例

```
communication normalReservation {
  MessageType {
    ResourceRequest,NO,OK,ACK,NG,Result
  }

  client( String air_no ){
    Object data;
    String No;

    start{
       next( state1 );
    }

    state1{
      receive( data ){
      case: ResourceRequest:
        No = (String)accessResource("air_no");
        send( NO, No );
        next( state2 );
      }
    }

    state2 {
      receive( data ){
        case OK:
          send( ACK, "OK" );
          next( state3 );
        case NG:
          send( ACK, "NG" );
          next( state3 );
      }
    }

    state3 {
      receive( data ){
      case Result:
        if ( data == "Success" ){
          result( "air_ID", No );
          next( end );
        }
        else if ( data == "Failure" ){
          next( end );
        }
      }
    }
  }

  server( ){
    Object data;
    String No;

    start{
      next( state1 );
    }

    state1{
      send( ResourceRequest, "No" );
      next( state2 );
    }

    state2{
      receive( data ){
      case No:
        No = ( String )data;
        next( state3 );
      }
    }

    state3{
      Boolean b1=callMethod(check(No));
      if ( b1 == Boolean.TRUE ){
        next( state4 );
      }
      else if ( b1 == Boolean.FALSE ){
        send( NG, "NG" );
        next( state5 );
      }
    }

    state4{
      Boolean b1=callMethod(reserve(No));
      send( OK, "OK" );
      next( state5 );
    }

    state5{
      receive( data ){
      case ACK:
        if ( data == "OK" ){
          send( Result, "Success" );
          next( end );
        }
        else if ( data == "NG" ){
          send( Result, "Failure" );
          next( end );
        }
      }
    }
  }
}
```

図 14: コミュニケーションズパッケージの記述例

```
public class A1 extends _apsl_MobileAgent {
  String     __apsl_i_air_host = "airplane.jp";
  String[ ]  __apsl_i_serial = {"123","136"};
  String[ ]  __apsl_i_air_ID = new String[2];
  String     __apsl_i_hotel_host = "hotel.jp";
  String     __apsl_i_checkin = "19970822";
  String     __apsl_i_checkout = "19970823";
  String     __apsl_i_hotel_ID;
  Boolean    __apsl_l_air;
  int        __apsl_l_i;
  int        __apsl_ll_i;
  int        __apsl_lll_i;

  public void main( String args[ ] ){
    A1 agent = new A1( );
    _apsl_move_to_place( agent );
  }
  public void _apsl_agent_init( ){
    _apsl_resource = new _apsl_AccessTable( );
    _apsl_register("__apsl_i_air_host",
                             __apsl_i_air_host,1);
    _apsl_register("__apsl_i_serial",
                             __apsl_i_serial,1);
    _apsl_register("__apsl_i_air_ID",
                             __apsl_i_air_ID,0);
    _apsl_register("__apsl_i_hotel_host",
                             __apsl_i_hotel_host,1);
    _apsl_register("__apsl_i_checkin",
                             __apsl_i_checkin,1);
    _apsl_register("__apsl_i_checkout",
                             __apsl_i_checkout,1);
    _apsl_register("__apsl_i_hotel_ID",
                             __apsl_i_hotel_ID,0);
  }
  public void _apsl_agen_run( ){
    try{
      switch( _apsl_state ){
      case 0:
        _apsl_move( _apsl_i_air_host );
        _apsl_step = 1;
        break;
      case 1 :
        for (__apsl_l_i=0;__apsl_l_i<2;__apsl_l_i++){
          try {
            _apsl_agent_init( );
            _apsl_useResource("__apsl_i_serial["
                              +__apsl_l_i+"]");
            _apsl_Result("__apsl_i_air_ID[
                        "+__apsl_l_i+"]",2);
            _apsl_communicate("RESERVATION");
          }catch(Exception e){e.printStackTrace();}
        }
        if ( air_ID[0]=null && air_ID[1]=null ){
          for (__apsl_ll_i=0;__apsl_ll_i<2;
                                     __apsl_ll_i++){
            try {
              _apsl_agent_init( );
              _apsl_useResource("__apsl_i_air_ID["
                                +__apsl_ll_i+"]",1);
              _apsl_communicate("CANCEL");
            }catch(Exception e){e.printStackTrace();}
          }
        }
        else if (air_ID[0]==null){
          try {
           _apsl_agent_init( );
           _apsl_useResource("__apsl_i_air_ID[1]",1);
           _apsl_communicate("CANCEL");
         }catch(Exception e){e.printStackTrace();}
       }
       else if (air_ID[1]==null){
         try {
           _apsl_agent_init( );
           _apsl_useResource("__apsl_i_air_ID[0]",1);
           _apsl_communicate("CANCEL");
         }catch(Exception e){e.printStackTrace();}
       }
       else { __apsl_l_air =  true; }
       if (__apsl_l_air==false){_apsl_state = 2;}
       else { _apsl_state = 4; }
       _apsl_agent_run( ); break;
     case 2 :
       _apsl_move(returnhost);_apsl_step=3;break;
     case 3 :
       System.out.println("Failure");break;
     case 4 :
       _apsl_move(_apsl_i_hotel_host);
       _apsl_step=5;break;
     case 5 :
       try {
         _apsl_agent_init( );
         _apsl_useResource("__apsl_i_checkin_date");
         _apsl_useResource("__apsl_i_checkout_date");
         _apsl_Result("__apsl_i_hotel_ID",2);
         _apsl_communicate("RESERVATION");
       }catch(Exception e){e.printStackTrace();}
       if (__apsl_i_hotel_ID==null){_apsl_state = 6;}
       else { _apsl_state = 9; }
       _apsl_agent_run( ); break;
     case 6 :
       _apsl_move(_apsl_i_air_host);_apsl_step=7;break;
     case 7 :
       for(__apsl_lll_i=0;__apsl_lll_i<2;
                                   __apsl_lll_i++){
         try {
           _apsl_agent_init( );
           _apsl_useResource( "__apsl_i_air_ID["
                              +__apsl_lll_i+"]",1);
          _apsl_communicate("CANCEL");
         }catch(Exception e){e.printStackTrace();}
       }
       _apsl_move(returnhost);_apsl_step=8;break;
     case 8 :
       System.out.println("Failure"); break;
     case 9 :
       _apsl_move(returnhost);_apsl_step=10;break;
     case 10 :
       System.out.println("Success"); break;
     }
    }catch(Exception e){e.printStackTrace();}
  }
}
```

図 15: APSL コンパイラによって生成された Java のプログラムの例