修士論文

題目

モーバイルエージェント記述言語に関する研究

指導教授

所 真理雄 客員教授
安西 祐一郎 教授

慶應義塾大学大学院

理工学研究科 計算機科学専攻

1997年度

89624206

勝野 恭治

論文要旨

専攻	計算機科学	学籍番号	89624206	氏名	勝野 恭治

(論文題目)

モーバイルエージェント記述言語に関する研究

モーバイルコンピュータを無線ネットワークを用いてインターネットに接続して作業を行なう際には，ネットワーク上を動き回るモーバイルエージェントを用いたプログラミングが有効な手段であると考えられている．また現在，モーバイルエージェントをJavaを用いて実装するのが適してい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Α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の記述が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．

本研究では，こ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)の提案，設計，及び実装を行う．APSLではJavaの構文の拡張を行い，モーバイルエージェントを容易に記述できるようにしている．そして，拡張した構文からJavaのコードを生成するAPSLコンパイラを提供している．また，通信プロトコルを動的に選択する機構を導入している．このためプログラマは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．本研究ではさらに性能評価を行ない，エージェントキャンパスガイドをアプリケーションとして実装することで，APSLの実用性を実証する．

慶應義塾大学理工学研究科修士課程

◎ 内容の要旨は約600字

Summary of Master's Thesis

Course	Student No.	SURNAME, Firstname
Dept. of Computer Science	89624206	KATSUNO, Yasuharu

Title

A Study on A Mobile Agent Programming Language

Mobile agents have been proposed as a programming model suited for mobile computers connected to the Internet by using wireless network. Java has been thought to be suitable for implementation of mobile agents. At present, There are some Java-based mobile agent programming languages.

However, there are some difficulties in programming mobile agents in terms of migration of mobile agents, communication among agents, and access control of resources that agents own. Therefore, programmers cannot program mobile agents simply.

To reduce the difficulties, we propose a Java-based mobile agent programming language named `APSL`(Agent Programming System and Language) in this paper along with its design and implementation. `APSL` extends syntax of Java for the purpose of programming mobile agents simply, and provides `APSL` compiler for the purpose of translating `APSL` source codes to Java source codes, and introduces a mechanism selecting a communications protocol dynamically. Therefore, `APSL` can reduce the programmer's load by its programmability. And, Performance evaluation and implementation of `AgentCampusGuide` prove applicability of `APSL`.

Graduate School of Science and Technology, KEIO UNIVERSITY

⊚ The contents of English summary should be written in about 300 words.

目次

図 目 次

表 目 次

第 1 章

序論

近年，ノートパソコン，PDA といったモーバイルコンピュータの小型化や携帯電話，PHS の普及に伴い，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て信頼性が低く，また通信速度が遅い．信頼性が低いネットワークは，作業中に突然ネットワークが切断される可能性が高い．また通信速度が遅いネットワークは，作業が終了するまでの時間が長い．そのため，モーバイルコンピュータをインターネットに接続して作業を行なう際には，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．モーバイルエージェントを用いるとユーザは，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．

モーバイルエージェントには以下のような特徴がある．

(a) ホスト間を移動し，移動先の各ホスト上で実行する．

(b) 様々なアーキテクチャ，OS 上で実行する．

これに対して，Java[9][10] には以下のような特徴がある．

(a) 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．

(b) インタプリタ言語なのでプログラムがアーキテクチャ，OS に依存しない．

以上より，Java はモーバイルエージェントの実装に適していると考えられている．現在，Aglet[2][3][4][5][20]，Java-To-Go[17]，Mole[23]，Java Agent Template[7], CyberAgent[15] 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 [21]．これらの言語は全て Java のライブラリとして実現されている．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，モーバイルエージェントを記述する際に次の 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
- エージェント間の通信

- 変数のアクセス制御

本研究では，モーバイルエージェントを記述する際のプログラマの負担を減らす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．さらに`APSL`の設計，実装，評価を行い，`APSL`のアプリケーションとしてエージェントキャンパスガイドを構築することで，`APSL`の実用性を実証する．

本論文は，本章も含めて8章から構成されている．第2章では，モーバイルエージェントについて説明した後，本研究の関連研究について説明する．第3章では，本研究の骨子であるモーバイルエージェント記述言語`APSL`の設計について説明する．第4章では，第3章で説明した`APSL`の実装方法を述べる．第5章では，第4章で実装したモーバイルエージェント記述言語`APSL`の評価を行う．第6章では，`APSL`のアプリケーション例として作成するエージェントキャンパスガイドについて説明し，エージェントキャンパスガイドの設計，実装について説明する．第7章では今後の課題，将来への展望などを検討し，第8章で本研究の結論を述べる．

第 2 章

背景

本章ではまずモーバイルエージェントについて説明する．続いて，関連研究として既存のモーバイルエージェント記述言語を紹介する．

2.1 モーバイルエージェント

モーバイルエージェントと一言でいっても，その定義は論文によって多少異なっている[2][3][4][5][7][15][17][20][21][23]．そこで本論文ではモーバイルエージェントを，ネットワーク上を動き回り，ネットワーク上の各ホストからサービスを獲得することでユーザ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は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ため，モーバイルコンピュータをインターネットに接続して作業を行う際に有効な手段であると考えられている．一般に，モーバイルコンピュータは有線ネットワークに比べて通信速度が遅く信頼性が低い無線ネットワークを用いてインターネットに接続する．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をインターネットに接続する時間を短縮したいという要望がある．しかし一般には，モーバイルコンピュータをインターネットに接続しインターネット上の仕事が終了するまでの間，モーバイルコンピュータをインターネットに接続し続けなければならない．そこで，モーバイルエージェントを使用すると，モーバイルコンピュータをインターネットに接続する時間は，モーバイルエージェントがモーバイルコンピュータからインターネットに移動する時と，作業を終えインターネットからモーバイルコンピュータに戻ってくる時のみとなる．即ちインターネット上でモーバイルエージェントが仕事を行っている間は，モーバイルコンピュータをインターネットに接続する必要はないのである．

これよりモーバイルエージェントが，モーバイルコンピュータをインターネットに接続して作業を行う際に有効な手段であるということが分かる．

2.2 関連研究

既存のモーバイルエージェント記述言語は大きく分けて，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 つに分けられる．本節では Java をベースとした言語の中から代表的な言語として Aglet[2][3][4][5][20]，Java-To-Go[17] を，Java をベースとしてない言語から代表的な言語として Telescript[8][26][27][28]，Agent Tcl[11][12][13][14][16] それぞれのエージェント記述の容易さについて検討する．

Aglet

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，Java のライブラリとしてモーバイルエージェントを実装している．Aglet は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という特徴がある．

Aglet は Java RMI ライブラリ [6][19] を使用してモーバイルエージェントの移動を実装している．移動の記述を図 2.1に示す．

```
try {
    switch ( _state ) {

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

    case HOST2:
        .....
        break;

    default:
        throw new AgletException("Error");
    }
} catch ( AgletException ae ) {
    System.out.println( ae.getMessage() );
    try {
       disposeThis();
    } catch ( Exception e ) {
       e.printStackTrace( );
    }
}
```

図 2.1: Aglet の移動の記述

Aglet はローカル変数の値やメソッドがどこまで進んだのかを示す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 2.1のように Java の `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を保持し，値を保持したいローカル変数をインスタンス変数に変換する必要がある．

また Aglet では，モーバイルエージェントはソケットを用いたメッセージの送受信によって他のエージェントと通信を行う．通信の記述を図 2.2に示す．

Aglet では，プログラマはメッセージを受け取る処理を `handleMessage` という名前の

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

図 2.2: Aglet の通信の記述

特別なメソッド内に記述しなければならない．そのため，通信プロトコルは固定している．また，メッセージを送信するエージェントとメッセージを受信するエージェントの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．

Java-To-Go

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，Java のライブラリとしてモーバイルエージェントを実装している．Java-To-Go は複数のエージェントに対してマルチキャスト通信を行うことができるという特徴がある．

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．しかし Aglet と同様に，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 と同様にように Java の `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を保持し，値を保持したいローカル変数をインスタンス変数に変換する必要がある (図 2.3)．

また Java-To-Go では，モーバイルエージェントはソケットを用いたメッセージの送受信によって他のエージェントと通信を行う (図 2.4)．

Java-To-Go では Aglet と同様に，メッセージを送信するエージェントとメッセージを受信するエージェントの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．しかし Java-To-Go は 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．そのため，プログラマは `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自由に変えることができる．しかし，通信プロトコルの変更を言語側で支援していない．

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

図 2.3: Java-To-Go の移動の記述

```
Message msg1;
.....
CommHandle ch = getCommHandle("getmail");
ch.write( msg1 );
.....
Message msg2 = ch.read( );
String str = ( String )msg2.content( );
```

図 2.4: Java-To-Go の通信の記述

Telescript

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 は初期のモーバイルエージェント記述言語として広く知られている．

Telescript は Aglet や Java-To-Go とは異なり，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る．移動の記述を図 2.5に示す．

```
.....
try {
    .....
    *.go( Ticket( nil, nil, host1 ) );
} catch e: Exception {
    exception = e;
}
```

図 2.5: Telescript の移動の記述

また Telescript では Aglet, Java-To-Go とは異なり，モーバイルエージェントはメソッド呼出しによって他のエージェントと通信を行なう．通信の記述を図 2.6に示す．

```
.....
*.enableEvents( PartEvent( clientName ) );
here@MeetingPlace.meet( Petition( clientName ) );
```

図 2.6: Telescript の通信の記述

Telescript ではメソッド呼出しによって通信を行なうため，通信相手のエージェントが持つメソッドの名前と引数の型を予め決めておかなければならない．また，通信プロトコルは固定されていないが，通信プロトコルの変更を言語側で支援していない．

Agent Tcl

Agent Tcl はダートマス大学で開発された言語で，Tcl[25] をベースとした言語である．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．

Agent Tcl では，仮想機械の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ることができる．そのため，モーバイルエージェントは移動先のホスト上で実行を再開することができる．Agent Tcl の移動の記述を図 2.7に示す．

```
set machines    "host1 host2 host3"
.....
foreach m $machines{
    if { [catch "agent_jump $m"] }{
        append list "$m:\nunable to JUMPto this machine\n\n"
    }else{
        set users [exec who]
        append list "$agent(local-server):\n$users\n\n"
    }
}
.....
```

図 2.7: Agent Tcl の移動の記述

Agent Tcl では，エージェント間の通信方法としてソケットを用いる方法とメソッド呼出しを用いる方法を用意している．図 2.8にソケットを用いる方法による通信の記述を示す．

Agent Tcl では，ソケットを用いて通信を行なう場合は Java-To-Go と，メソッド呼出しを用いて通信を行なう場合は Telescript と同様な問題がそれぞれ生じてしまう．

```
.....
set sockfd [agent_meet "$agent(local-ip) file_agent"]
tcpip_write $sockfd "{NEW_MESSAGES 0} {IN_BOX_MESSAGES 10}"
.....
set status [tcpip_read $sockfd -blocking]
tcpip_close $sockfd
.....
```

図 2.8: Agent Tcl の通信の記述

2.3 まとめ

本章ではまず，モーバイルエージェントについて説明し，モーバイルエージェントがモーバイルコンピュータをインターネットに接続して作業を行う際に有効な手法であるということを説明した．続いて，関連研究として既存のモーバイルエージェント記述言語を紹介した．

3 章では，既存のモーバイルエージェント記述言語の問題点を指摘し，その問題点を解決する `APSL` について詳しく見ていく．

第 3 章

APSL の設計

本章ではまず既存のモーバイルエージェント記述言語の問題点について説明し，モーバイルエージェント記述言語 `APSL` の目的について説明する．続いて，`APSL` のシステムモデル，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設計についてそれぞれ説明する．そして最後に，各要素の記述構文，記述例を説明する．

3.1 既存のモーバイルエージェント記述言語の問題点

インターネットを構成するホストのアーキテクチャ，OS は様々である．そのため，モーバイルエージェント記述言語はアーキテクチャ，OS に依存しないインタプリタ言語の方が望ましく，また言語の仮想機械が様々なアーキテクチャ，OS に移植されている方が望ましい．また，モーバイルエージェントはホスト間を移動し，移動先のホストで実行する．そのため，モーバイルエージェント記述言語はプログラムを他のホストに移動させ，移動先のホストで実行させる機能を持っていなければいけない．それに対して，Java はインタプリタ言語であり，現在 Java の仮想機械は様々なアーキテクチャ，OS に移植されている．また，Java はオブジェクトを他のホストに移動させ，移動先のホストで実行させることができる．そのため，Java はモーバイルエージェントの実装に適していると考えられている．

現在 Aglet[2][3][4][5][20],Java-To-Go[17],Mole[23],JavaAgentTemplate[7],CyberAgent[15] 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 [21]．これらは全て Java のライブラリとして実現されているので，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，図 2.1, 2.3 のように Java の `if` 文や `switch` 文と実行カウンタを示す変数を使用して実行カウンタを保持し，値を保持したいローカル変数をインスタンス変数に変換する必要がある．

- エージェント間の通信

  エージェント間の通信は一般に図 3.1のような状態遷移図で表現される．そして，通信を行なうエージェントは状態遷移図，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 (図 2.4)．そのため，プログラマは通信で使用する通信プロトコルと各通信で用いられるメッセージフォーマットを事前に知らなければならない．しかしメッセージフォーマット，通信プロトコルは同じサービスでもサービスを提供するホストによって異なる．さらに通信プロトコルは，同じホストの同じサービスでも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，プログラマが状況に最適な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事前に知るのは大変困難である．

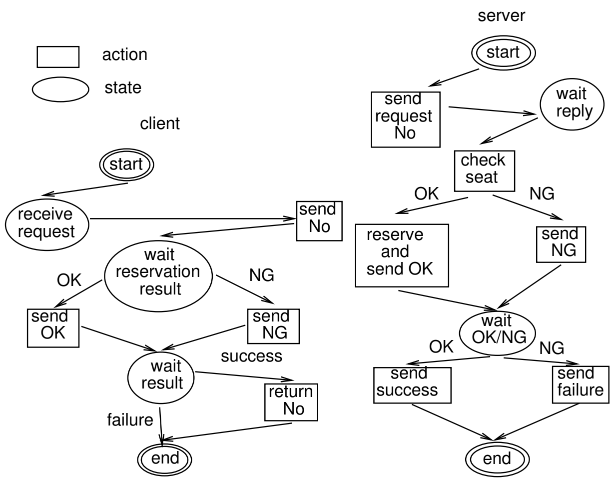


図 3.1: 通信プロトコルの例

- 変数のアクセス制御

  モーバイルエージェントは，ソケット通信によって他のエージェントの変数にアクセスする．そのため，他のエージェントに知られたくない情報を保持している変数のアクセス制御がたとえ `private` になっていても，ソケットを通じて他のエージェントにアクセスされてしまう恐れがある．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い．

3.2 APSL の目的

本研究で提案するモーバイルエージェント記述言語 `APSL`(Agent Programming System and Langauge) は，モーバイルエージェントを記述する際のプログラマの負担を減らすことを目的としている．`APSL` では次の 3 点を行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 Java の構文を拡張して，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．

- `APSL` コンパイラの提供

  `APSL` 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．`APSL` コンパイラによって Java の仮想機械を修正する必要がなくなるので，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て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．

3.3 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 3 つの要素から構成されている (図 3.2)．

- 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．

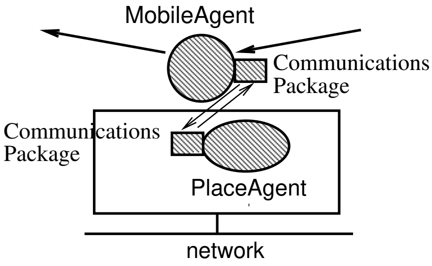


図 3.2: APSL の構成要素

- 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通信の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起動すると，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エージェントが持つメソッドを呼び出したりすることができる．

本研究では，APSL が用いられるネットワーク環境として図 3.3のようなネットワーク環境を考えている．

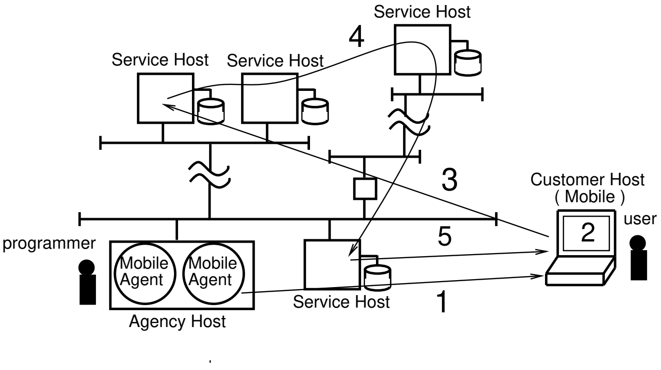


図 3.3: APSL が用いられるネットワーク環境

APSL が用いられるネットワーク環境には，エージェンシーホスト，サービスホスト，カスタマーホストという 3 種類のホストが存在する．

- エージェンシーホスト

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ホスト

- サービスホスト

  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ホスト

- カスタマーホスト

  モーバイルエージェントを用いるユーザがいるホスト．常にネットワークに接続されているとは限らない．

図 3.3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．

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．この時点で，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ことができる．
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．

3.4 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する．そのため，モーバイルエージェントは移動先のホストで継続した実行を再開することができる．

3.5 エージェント間の通信

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．またプレースエージェントは，モーバイルエージェント用，プレースエージェント用の両方のコミュニケーションズパッケージを持っており，エージェント間で通信を行う際に，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す．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，それを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となる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がその通信で用いることが許可されているかどうか調べる．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，それを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存在しなければエラーメッセージを送る．

以上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，エージェントの持つ変数へのアクセスやメソッドの呼び出しを行なう際の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とアクセスの種類，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許可状態を調べる．
3. もしアクセスが許可されていれば，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．もしアクセスが許可されていなければ，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．

3.6　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許可状態を管理するテーブルを持っている．エージェントは他のエージェントから変数へのアクセスを要求された時点で，そのテーブルを用いて該当する変数のアクセス許可状態を調べる．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し，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許可状態を以下に示す．

- read，write アクセスを共に許可しない
- read アクセスを許可する
- write アクセスを許可する
- read,write アクセスを共に許可する

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

3.7　記述構文

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の記述構文は Java の記述構文とほぼ同じである．そこで本節では，Java の記述構文の拡張部分について説明する．[1]

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の記述構文を説明する前に，まずそれぞれに共通な Java の記述構文との相違点を以下に示す．

- クラス宣言ができない
- クラス名，メソッド名，変数名の前に_apsl_, __apsl_ ,... をつけることができない．

3.7.1　モーバイルエージェントの記述構文

モーバイルエージェントの記述例を以下に示す．

---

[1] 記述構文の詳細は付録 A に記してある

```
agent AGENTNAME {
    String:[rw]: str1;
    .....
    main( ){
        .....
    }

    incomm Object method1( Vector vec ){
        .....
    }
}
```

プログラマはモーバイルエージェントを記述する際に，移動先のホストで実行を再開する際に呼び出される `main` メソッドを必ず記述しなければならない．また，プログラマは上記の `method1` メソッドのように `main` メソッド以外のメソッドを記述することができる．`main` メソッド以外のメソッドと Java のメソッドとの違いを以下に述べる．

- 返り値の前にコミュニケーションズパッケージからの呼び出しを許可するかどうかを表す `incomm`，または `nocomm` をつける．`incomm` は呼び出しを許可することを表し，`nocomm` は呼び出しを許可しないことを表す．
- `incomm` がついているメソッドの返り値は必ず `Object` クラスから継承されたクラスでなくてはならない．また，引数は必ず `Vector` クラスでなくてはならない．

モーバイルエージェントは疑似変数として，モーバイルエージェントの識別子を表す `agent_id` と，全ての作業が終了した時に移動するサービスホストのホスト名を表す `returnhost` という変数を持っている．モーバイルエージェントを実際に用いる際には，`agent_id` と `returnhost` を必ず初期化しなければならない．

ホスト間の移動

ホスト `HOSTNAME` に移動し，そのホスト上で中括弧で囲まれている部分に記述されている作業を行うモーバイルエージェントは以下のように記述される．

```
host( String HOSTNAME ){
    .....
}
```

現時点の APSL では，ホスト間の移動は `main` メソッド内のみしか記述することができず，また入れ子にして記述することができない．

エージェント間の通信

サービス`SERVICE_ NAME`を得るために，引数として変数`RESOURCE_NAME1`，`RESOURCE_NAME2`，`etc` を使用し，通信の結果を変数 `RETURN_ VAR1`, `RETURN_VAR2`，`etc` に格納するモーバイルエージェントは以下の様に記述される．

```
communicate( SERVICE_NAME ){
    RESOURCE_NAME1,
    RESOURCE_NAME2,
    .....
} reply {
    RETURN_VAR1,
    RETURN_VAR2,
    .....
}
```

現時点の`APSL`では，`RESOURCE_NAME1`，`RESOURCE_NAME2`，`etc` や`RETURN_ VAR1`, `RETURN_VAR2`，`etc` に記述することができる変数はインスタンス変数のみである．

変数のアクセス制御

現時点の`APSL`では，アクセスを制御することができる変数はインスタンス変数のみであり，変数のアクセス制御の宣言は変数を宣言する際に行われる．クラス名が`CLASS_NAME`，アクセス許可状態が`ACCESS_ PERMISSION`，変数名が`VARIABLE_NAME`という変数は以下の様に記述される．

```
CLASS_NAME ACCESS_PERMISSION  VARIABLE_NAME;
```

:[-]:	read，write アクセスを共に許可しない
:[r]:	read アクセスを許可する
:[w]:	write アクセスを許可する
:[rw]:	read,write アクセスを共に許可する

表 3.1: 変数のアクセス許可状態の構文

`ACCESS_PERMISSION`の構文を表 3.1に示す．`ACCESS_PERMISSION`の記述は省略することができ，アクセス許可状態が記述されていない変数のアクセス許可状態は`:[r]:`になる．

また，プログラマはエージェント間の通信の記述中で，変数のアクセス許可状態を一時的に変更することができる．変数`VARIABLE_NAME`のアクセス許可状態を`TEMPORAL_PERMISSION`に変更するモーバイルエージェントは以下の様に記述される．

```
TEMPORAL_PERMISSION VARIABLE_NAME
```

アクセス許可状態を一時的に変更された変数のアクセス許可状態は，通信が終了すると元のアクセス許可状態に戻る．

3.7.2 プレースエージェントの記述構文

プレースエージェントの記述例を以下に示す．

```
place PLACENAME {
    init( ){
        .....
    }

    communication( ){
        .....
    }

    nocomm void method1( ){
        .....
    }
}
```

プログラマはプレースエージェントを記述する際に，プレースエージェントが実行された時に自動的に呼び出される `init` メソッドと，モーバイルエージェントからの通信の要求を受け取った時に自動的に呼び出される `communication` メソッドを必ず記述しなければならない．また，プログラマはモーバイルエージェントと同様に `init` メソッド，`communication` メソッド以外のメソッドを宣言することができる．`init` メソッド，`communication` メソッド以外のメソッドと Java のメソッドとの違いはモーバイルエージェントの場合と同様である．

エージェント間の通信

通信で用いるコミュニケーションズパッケージの登録は `init` メソッド内に，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と通信を行う部分は `communication` メソッド内にそれぞれ記述される．

`APSL` では `registCommunication` メソッドというコミュニケーションズパッケージを登録するメソッドを用意している．構文を以下に示す．

- `void registCommunication(String RNAME, Communication CPNAME );`

  コミュニケーションズパッケージ `CPNAME` を `RNAME` という名前で登録する．

また，コミュニケーションズパッケージを選択し，選択したコミュニケーションズパッケージを用いてモーバイルエージェントと通信を行うプレースエージェントは以下の様に記述される．

```
void communication( ){
    service( SERVICE1 ){

    case[ RESOURCE1, RESOURCE2 ]:
        start( "COMMPKG1" );

    case[ RESOURCE2 ]:
        start( "COMMPKG2" );

    default:
        comm_error( );
    }

    service( SERVICE2 ){
       .............
    }
}
```

上記のプログラムの実行手順を説明する．

1. モーバイルエージェントが要求するサービスがもし `SERVICE1` だったら，プレースエージェントはモーバイルエージェントが変数 *RESOURCE1* と *RESOURCE2* を通信で使用できるかどうかを調べる．
2. (a) もしモーバイルエージェントが変数 `RESOURCE1` と `RESOURCE2` を使用することができれば，プレースエージェントは `COMMPKG1` という名前で登録されているコミュニケーションズパッケージを選択して通信を行う．
   (b) もしモーバイルエージェントが変数 `RESOURCE2` しか使用することができなければ，プレースエージェントは `COMMPKG2` という名前で登録されているコミュニケーションズパッケージを選択して通信を行う．
   (c) もしモーバイルエージェントが変数 `RESOURCE2` を使用することができなければ，プレースエージェントはモーバイルエージェントにエラーメッセージを送る．

`APSL` では次のメソッドを用いてモーバイルエージェントとの通信を記述する．

- `void start( String COMMPKGNAME );`

  COMMPKGNAME という名前で登録されているコミュニケーションズパッケージを使用して，モーバイルエージェントと通信を行う．

- `void comm_error( );`

  エラーメッセージをモーバイルエージェントに送る．

変数のアクセス制御

モーバイルエージェントと同様．

3.7.3 コミュニケーションズパッケージの記述構文

コミュニケーションズパッケージの記述例を以下に示す．

```
communication COMMPKGNAME {
    MessageType {
        ........
    }

    client( ... ){
        ........
    }

    server( ... ){
        ........
    }
}
```

コミュニケーションズパッケージは，メッセージタイプ部分，モーバイルエージェント側の通信部分，プレースエージェント側の通信部分という 3 つの部分から構成されている．そしてプログラマは，コミュニケーションズパッケージ内にメソッドやインスタンス変数を記述できない．

メッセージタイプ部分

メッセージタイプとは，通信の送受信時に用いられるメッセージの識別子である．通信で使用するメッセージタイプは `MessageType{ .... }` 内に全て記述しなければならない．メッセージタイプ部分の記述例を以下に示す．

```
MessageType {
    MT1, MT2, MT3
}
```

上記のプログラム中の `MT1`, `MT2`, `MT3` は，通信で用いるメッセージタイプである．

モーバイルエージェント側の通信部分

モーバイルエージェント側の通信部分の記述を以下に示す．

```
client( CLASSNAME1 VARNAME1, CLASSNAME2 VARNAME2,... ){
    start{
        .....
        next( state1 );
    }
    state1{

        .....
    }
    .....
    state7{
        .....
        next( end );
    }
}
```

`CLASSNAME1 VARNAME1, CLASSNAME2 VARNAME2,...` には，モーバイルエージェントが通信で用いる変数のクラス名と変数名を記述する．そして，`APSL` では次のメソッドを用いてモーバイルエージェント側の通信部分を記述する．

- *STATE_NAME* { ..... }

  状態遷移図内の *STATE_NAME* という名前のステートを表す．ステート内で行う仕事を中括弧の中に記述する．ステートは必ず `begin` で始まり，`end` で終了しなければならない．

- `next(` *NEXT_STATE*);

  NEXT_ STATE という名前のステートに遷移する．

- `result(` *String REGISTNAME, Object VARNAME*);

  変数 *VARNAME* の値を *REGISTNAME* という名前の変数に格納する．そして，通信が終了した時にその変数をモーバイルエージェントに渡す．

- *Object* `callMethod`(*METHODNAME*(*Object VARNAME*));

  モーバイルエージェントが持つ *METHODNAME* メソッドを引数 *VARNAME* で呼び出す．呼び出すことができるメソッドは，モーバイルエージェント側で `incomm` と記述されているメソッドのみである．

- *Object* `resourceAccess`(*String VARNAME*);

  モーバイルエージェントが通信で用いる変数 *VARNAME* に read アクセスを行う．

- `send`(*MESSAGETYPE*, *Object VARNAME*);

  メッセージタイプ *MESSAGETYPE* と変数 *VARNAME* の値をメッセージに添付し，そのメッセージをプレースエージェントに送る．

- `receive`(*Object OBJECTNAME*){

  `case` *MESSAGETYPE1*:

  .....

  `break`;

  `case` *MESSAGETYPE2*:

  .....

  `break`;

  `case` *MESSAGETYPE3*:

  .....

  `break`;

  }

  プレースエージェントから送られてきたメッセージを受け取り，メッセージに添付されているオブジェクトの値を *OBJECTNAME* に格納し，メッセージのメッセージタイプに応じた処理を行う．*OBJECTNAME* のクラス名は `Object` でなければならない．

プレースエージェント側の通信部分

モーバイルエージェント側の通信部分と同様．

3.8 記述例

飛行機のチケットとホテル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付けるプレースエージェント，図 3.1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 3.4, 3.5, 3.6 に示す．

図 3.4のモーバイルエージェントは，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 (15, 35 行目)，キャンセルする時には read アクセスのみを許可し (19, 41 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)．

図 3.5のプレースエージェントは，`RESERVATION` 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中から，`CANCEL` サービスを提供する際には 1 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許可されている変数の状況に応じてそれぞれ選択している (10–25 行目)．また，`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 (27, 29, 31 行目)．

図 3.4, 3.5より，図 3.4のモーバイルエージェントが図 3.5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．

最後に図 3.6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している (71, 82 行目)．

```
agent A1 {
  String air_host = "airplane.jp";
  String serial = "123";
  String:[-]: air_ID;
  String hotel_host = "hotel.jp";
  String checkin = "19970822";
  String checkout = "19970823";
  String:[-]: hotel_ID;
  main(  ){
    boolean air;
    host( air_host ){
      communicate( RESERVATION ){
        serial
      }  reply  {
        :[w]:air_ID
      }
      if ( air_ID == null ){
        communicate( CANCEL ){
          :[r]:air_ID
        } 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 ){
          communicate( CANCEL ){
            :[r]:air_ID[ i ]
          }  reply  {  }
        }
        host( returnhost ){
          System.out.println("Failure"); }
      }
      else {
        host( returnhost ){
          System.out.println("Success"); }
      }
    }
  }
}
```

図 3.4: モーバイルエージェントの記述例

```
place P1 {
  DB db = new DB( );
  init( ){
    registCommunication("normal",Normal);
    registCommunication("member",Member);
    registCommunication("vip",   Vip);
    registCommunication("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 ]:
        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 3.5: プレースエージェントの記述例

```
communication Normal {
  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 ( (String)data == "Success" ){
          result( "air_ID", No );
          next( end );
        }
        else if ( (String)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 3.6: コミュニケーションズパッケージの記述例

3.9 まとめ

本章ではまず既存のモーバイルエージェント記述言語の問題点について説明し，モーバイルエージェント記述言語 APSL の目的について説明した．続いて，APSL のシステムモデル，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設計について説明した．そして最後に，各要素の記述構文，記述例を説明した．

第 4章では，APSL の実装について詳しく述べる．

第 4 章

APSL の実装

本章ではまず APSL の実行の手順を説明し，続いて APSL コンパイラ，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実装方法について説明する．

4.1 APSL の実行の手順

本論文では，APSL のソースプログラムから Java のソースプログラムを生成する APSL コンパイラと APSL のソースプログラムから作成された Java の中間コードをリンクする際に用いられる APSL ランタイムライブラリ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を図 4.1を用いて説明する．

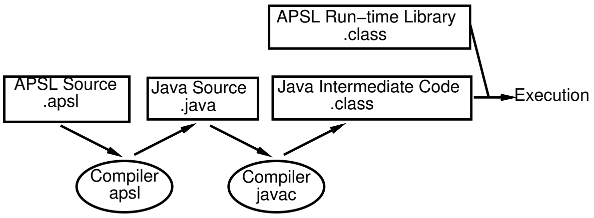


図 4.1: APSL の実行の手順

1. APSL のソースプログラムは APSL コンパイラによって Java のソースプログラムに変換される．
2. Java のソースプログラムは JDK(Java Developers Kit) で提供されている javac コンパイラによって Java の中間コードに変換される．
3. 生成した Java の中間コードは JDK で提供されている Java の仮想機械の上で APSL ランタイムライブラリとリンクされ，実行される．

4.2 APSL コンパイラ

本論文では構文解析部分に bison version 1.25 を，意味解析部分に flex version 2.5.4 を，コード生成部分に C++(gcc version 2.7.2.1) をそれぞれ用いて，APSL コンパイラを実装した．コンパイルの手順を以下に示す．

1. APSL のソースを解析し，構文の各要素に対応するノードを作成する．作成された各ノードは図 4.2のような木構造を構成している．各ノードは `FirstCompile` メソッドと `SecondCompile` メソッドを持っている．

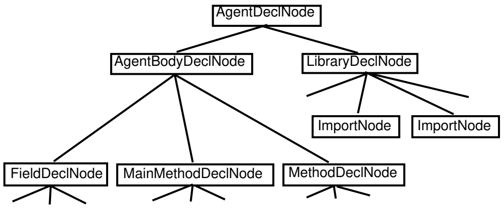


図 4.2: APSL コンパイラによって作成されるノード

2. 木構造のトップのノードの `FirstCompile` メソッドを実行する．すると，それ以下のノードの `FirstCompile` メソッドが順々に実行される．`FirstCompile` メソッドでは `SecondCompile` を行う際に用いられるテーブルを作成する．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で作成するテーブルを以下に示す．
   - モーバイルエージェント
     - インスタンス変数名の一覧表
     - `main` メソッド内のローカル変数のクラス名と名前の一覧表
     - コミュニケーションズパッケージから呼び出されるメソッド名の一覧表
   - プレースエージェント
     - サービスを提供する際にチェックするモーバイルエージェントの変数名の一覧表
     - 登録するコミュニケーションズパッケージ名の一覧表
     - コミュニケーションズパッケージから呼び出されるメソッド名の一覧表
   - コミュニケーションズパッケージ
     - 使用するメッセージタイプの一覧表
3. 木構造のトップのノードの `SecondCompile` メソッドを実行する．すると，それ以下のノードの `SecondCompile` メソッドが順々に実行され，それぞれ各ノードは Java のソースを生成する．

図 3.4, 3.5, 3.6 の `APSL` のソースプログラムは `APSL` コンパイラによってそれぞれ図 4.3, 4.4, 4.5 の Java のソースプログラムに変換される．

```
public class A1 extends _apsl_MobileAgent{
  private String __apsl_i_air_host="airplane.jp";
  private String __apsl_i_serial = "123";
  private String __apsl_i_air_ID;
  private String __apsl_i_hotel_host = "hotel.jp";
  private String __apsl_i_checkin = "19970822";
  private String __apsl_i_checkout = "19970823";
  private String __apsl_i_hotel_ID;
  private boolean __apsl_l_air;

  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);
   _apsl_register("__apsl_l_air",
                           __apsl_l_air);
  }
  public void _apsl_agen_run( ){
    try{
    switch( _apsl_state ){
    case 0:
      _apsl_move( __apsl_i_air_host );
      _apsl_step = 1;
      break;
    case 1 :
      try {
      _apsl_agent_init( );
      _apsl_Use("__apsl_i_serial");
      _apsl_Result("__apsl_i_air_ID",2);
      _apsl_communicate("RESERVATION");
      }catch( Exception e ){ }
      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_Use("__apsl_i_checkin");
      _apsl_Use("__apsl_i_checkout");
      _apsl_Result("__apsl_i_hotel_ID",2);
      _apsl_communicate("RESERVATION");
      } catch ( Exception e ){ }
      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 :
      try {
      _apsl_agent_init( );
      _apsl_Use("__apsl_i_air_ID",1);
      _apsl_communicate("CANCEL");
      } catch ( Exception e ){ }
      _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 4.3: APSL コンパイラによって生成されたモーバイルエージェントの Java プログラムの例

```
public class P1 extends _apsl_PlaceAgent {
  DB db = new DB( );
  _apsl_cpc_normalReserve _apsl_cpc_normalReserve
       = new _apsl_cpc_normalReserve( );
  _apsl_cps_normalReserve _apsl_cps_normalReserve
       = new _apsl_cps_normalReserve( );
  _apsl_Communication _apsl_normalReserve
       = new _apsl_Communication(
          _apsl_cpc_normalReserve,
          _apsl_cps_normalReserve );
 _apsl_cpc_membersReserve _apsl_cpc_membersReserve
       = new _apsl_cpc_membersReserve( );
 _apsl_cps_membersReserve _apsl_cps_membersReserve
       = new _apsl_cps_membersReserve( );
  _apsl_Communication _apsl_membersReserve
       = new _apsl_Communication(
          _apsl_cpc_membersReserve,
          _apsl_cps_membersReserve );
  _apsl_cpc_vipReserve _apsl_cpc_vipReserve
       = new _apsl_cpc_vipReserve( );
  _apsl_cps_vipReserve _apsl_cps_vipReserve
       = new _apsl_cps_vipReserve( );
  _apsl_Communication _apsl_vipReserve
       = new _apsl_Communication(
          _apsl_cpc_vipReserve,
          _apsl_cps_vipReserve );
  p1( ){ super( ); _apsl_register( );  }
  void _apsl_register( ){
    _apsl_registCommunication( "normal",
                                normalReserve );
    _apsl_registCommunication( "member",
                                memberReserve );
    _apsl_registCommunication( "vip",
                                vipReserve );
  }
  void _apsl_communication( String service ){
    int i = 0;
    if((i=service.compareTo("RESERVATION"))==0){
      _apsl_checkInit("serial");
      _apsl_checkInit("member");
      _apsl_checkInit("vip");
      _apsl_check( );
      boolean _apsl_b_serial =
                  _apsl_checkResult("serial");
      boolean _apsl_b_member =
                  _apsl_checkResult("member");
      boolean _apsl_b_vip =
                     _apsl_checkResult("vip");
      if(_apsl_b_serial==true&&_apsl_b_vip==true){
        _apsl_start("vip");
      }
      if(_apsl_b_serial==true &&
                          _apsl_b_member==true){
        _apsl_start("member");
      }
      if ( _apsl_b_serial == true ){
        _apsl_start("normal");
      }
      else {
        _apsl_comm_error( );
      }
    }
    else if((i=service.compareTo("CANCEL"))==0){
      _apsl_checkInit("air_ID");
      _apsl_check( );

      boolean _apsl_b_air_ID =
                     _apsl_checkResult("air_ID");

      if ( _apsl_b_air_ID==true ){
        _apsl_start("cancel");
      }
      else {
        _apsl_comm_error( );
      }
    }
  }
  public Object _apsl_callMethod( Vector vec ){
    String name = ( String )vec.elementAt( 0 );
    int cmp;
    if ( ( cmp = name.compareTo("check") ) == 0 ){
      return( check( vec ) );
    }
    if ( ( cmp = name.compareTo("reserve") ) == 0 ){
      return( reserve( vec ) );
    }
    if ( ( cmp = name.compareTo("cancel") ) == 0 ){
      return( cancel( vec ) );
    }
    return( null );
  }
  private Boolean seatCheck( Vector vec ){
    return( db.check( vec ) );
  }
  private Boolean reserve( Vector vec ){
    return( db.reserve( vec ) );
  }
  private Boolean cancel( Vector vec ){
    return( db.cancel( vec ) );
  }
}
```

図 4.4: APSL コンパイラによって生成されたプレースエージェントの Java プログラムの例

```
class _apsl_cpc_normalReserve
                    extends _apsl_ClientCommPkg {
  static final int ResourceRequest = 0;
  static final int NO              = 1;
  static final int OK              = 2;
  static final int Selection       = 3;
  static final int ACK             = 4;
  static final int Another         = 5;
  static final int NG              = 6;
  static final int Result          = 7;
  Object data;  String No;
  void __apsl_begin( ){ __apsl_state1( ); }
  void __apsl_state1( ){
    _apsl_Message msg = _apsl_receive( );
    int type = msg.MessageType;
    data = msg.data;
    switch( type ){
    case ResourceRequest:
      No=(String)_apsl_resourceAccess("air_no");
      _apsl_send(No,No);__apsl_state2( );break;}
  }
  void __apsl_state2( ){
    _apsl_Message msg = _apsl_receive( );
    int type = msg.MessageType;
    data = msg.data; switch( type ){
    case OK:
      _apsl_send( ACK, "OK" );
      __apsl_state3( ); break;
    case NG:
      _apsl_send( ACK, "NG" );
      __apsl_state3( ); break; }
  }
  void __apsl_state3( ){
    _apsl_Message msg = _apsl_receive( );
    int type = msg.MessageType;
    data = msg.data; switch( type ){
    case Result:
      if ( ( String )data == "Sucess" ){
        _apsl_result( "air_ID", No );
        __apsl_end( ); break;
      }
      else if ( ( String )data == "Failure" ){
        _apsl_result( "air_ID", null );
        __apsl_end( ); break;
      }    }  }
}
class _apsl_cps_normalReserve
                    extends _apsl_ServerCommPkg {
  static final int ResourceRequest = 0;
  static final int NO              = 1;
  static final int OK              = 2;
  static final int Selection       = 3;
  static final int ACK             = 4;
  static final int Another         = 5;
  static final int NG              = 6;
  static final int Result          = 7;
  Object data;  String No;
  void __apsl_begin( ){ __apsl_state1( ); }
  void __apsl_state1( ){
    _apsl_send( ResourceRequest, "No" );
    __apsl_state2( );
  }
  void __apsl_state2( ){
    _apsl_Message msg = _apsl_receive( );
    int type = msg.MessageType;
    data = msg.data; switch( type ){
    case No:
      No = ( String )data;
      __apsl_state3( ); break; }
  }
  void __apsl_state3( ){
    Boolean b1 = ( Boolean )_apsl_callMethod(
                    "seatCheck", No );
    if ( b1 == Boolean.TRUE ){
      __apsl_state4( );  }
    else if ( b1 == Boolean.FALSE ){
      _apsl_send( NG, "NG" );
      __apsl_state5( ); }
  }
  void __apsl_state4( ){
    Boolean b1 = ( Boolean )_apsl_callMethod(
                   "reservationSeat", No );
    _apsl_send( OK, "OK" );  __apsl_state5( );
  }
  void __apsl_state5( ){
    _apsl_Message msg = _apsl_receive( );
    int type = msg.MessageType;
    data = msg.data; switch( type ){
    case ACK:
      if ( ( String )data == "OK" ){
        _apsl_send( Result, "Success" );
        _apsl_end( ); break;
      }
      else if ( ( String )data == "NG" ){
        _apsl_send( Result, "Failure" );
        _apsl_end( ); break;
      }    }  }
}
public class _apsl_normalReserve
                    extends _apsl_Communication {
  _apsl_cpc_normalReserve _apsl_cpc_normalReserve;
  _apsl_cps_normalReserve _apsl_cps_normalReserve;
  _apsl_normalReserve( ){
   _apsl_cpc_normalReserve=new _apsl_cpc_normalReserve();
   _apsl_cps_normalReserve=new _apsl_cps_normalReserve();
  }}
```

図 4.5: APSL コンパイラによって生成されたコミュニケーションズパッケージの Java プログラムの例

4.3 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 ライブラリをそのまま用いただけでは以下の 2 つの問題があり，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ない．

(a) オブジェクトのローカル変数の値を保持することができない．

(b) オブジェクトの実行カウンタの値を保持することができない．

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値のみをシリアライズしているからである．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．そして変数名の衝突を防ぐため，インスタンス変数だった変数には`__apsl_i_`を，ローカル変数だった変数には`__apsl_l_`を変数名の前にそれぞれつける．[1]

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

続いて，ホスト間の移動の機構を図 4.6を用いて説明する．

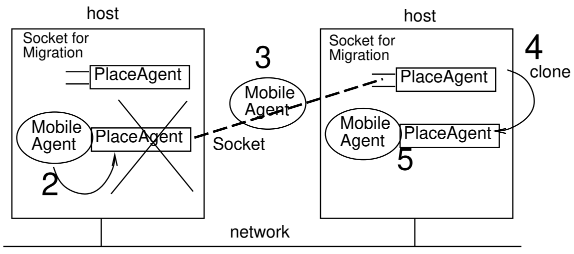


図 4.6: ホスト間の移動の機構

0. 各ホスト上には well-known ポート[2]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クローンのプレースエージェントの上にいる．

---

[1] 同じ名前のローカル変数が複数ある場合には出現回数に応じて__apsl_ll_，__apsl_lll_ ... をつける

[2] 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自分が今いるクローンの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クローンのプレースエージェントは移動先のホスト上のプレースエージェントにソケットの作成を要求する．そしてソケットが作成されると，クローンのプレースエージェントはそのソケットを用いてモーバイルエージェントを移動させる．クローンの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．
3. 移動先のホスト上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り，自分のクローンを作成する．そしてクローンのプレースエージェントにモーバイルエージェントを渡す．
4. 移動先のホスト上のクローンのプレースエージェントは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．

4.4 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，ソケットを用いてエージェントが持つ変数にアクセスしたりエージェントが持つメソッドを呼び出したりする場合がある (図 3.1, 3.6)．これを実現するため `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独立したスレッドとして実行させ，ソケットを持たせた (図 4.7)．

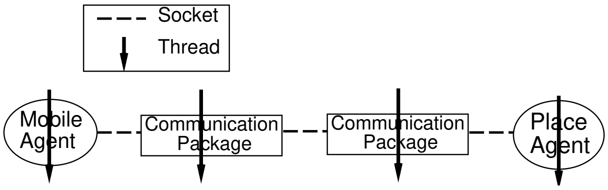


図 4.7: 通信の機構

4.5 変数のアクセス制御

`APSL` では`_apsl_AccessTable` クラスと`_apsl_Variable` クラスを実装して，変数のアクセス制御を実現した．`_apsl_Variable` クラスはインスタンス変数の名前，値，アクセス

許可状態を保持している．`_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 4.8に示す．

```
 1 public class _apsl_Variable extends Object
 2         implements Serializable, Cloneable {
 3     public String  name;
 4     public Object  value;
 5     public int     access;
 6
 7     Variable( ..... ){
 8         .....
 9     }
10     .....
11     .....
12 }
```

図 4.8: _apsl_Varible クラス

図 4.8の 3 行目の `name` は変数の名前，4 行目の `value` は変数に格納されている値，5 行目の `access` は変数のアクセス許可状態をそれぞれ示している．`access` の値とその値に対するアクセス許可状態を表 4.1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

表 4.1: access フィールドの値と値に対するアクセス状態

4.6 まとめ

本章ではまず APSL の実行の手順を説明し，続いて APSL コンパイラ，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実装方法について説明した．

第5章では，本章で実装した APSL の評価を行う．

第 5 章

評価及び検討

評価は全て PC/AT 互換機 (CPU: IntelPentium 166Mhz, メモリ: 64M) で動作している Solaris 2.5.1 for x86 上で行った．そして実行時間の測定には，`APSL` コンパイラの評価では csh に組み込まれている `time` コマンドを，`APSL` ランタイムライブラリの評価では Java の `System` クラスの `currentTimeMillions` メソッドをそれぞれ用いた．

5.1 APSL のファイルサイズと Java のファイルサイズとの比較

本節では，`(a)` モーバイルエージェント (図 3.4)，`(b)` プレースエージェント (図 3.5)，`(c)` コミュニケーションズパッケージ (図 3.6) それぞれの `APSL` ファイルと `APSL` コンパイラによって生成された Java ファイル (図 4.3, 4.4, 4.5) のファイルサイズを測定した．測定結果を表 5.1に示す (単位 byte)．

(a)		(b)		(c)	
APSL	Java	APSL	Java	APSL	Java
1171	2535	856	2319	1711	3850

表 5.1: ファイルサイズの比較

表 5.1より，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の Java ファイルのファイルサイズはそれぞれ `APSL` ファイルの 2.16 倍，2.71 倍，2.25 倍になった．これより `APSL` が約 59.15%プログラムの記述量を減らすことが分かった．

5.2 APSL コンパイラのコンパイルスピードの測定

本節では，`APSL` ファイルをコンパイルするのにかかる時間を `APSL` ファイルのサイズを変えて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について測定した．測定結果を図 5.1に示す．

図 5.1より，ファイルサイズとコンパイル時間が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共にほぼ比例し，規模拡張性があることが分かった．

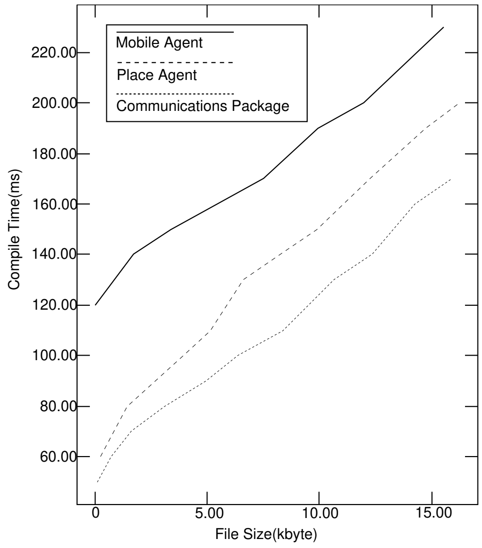


図 5.1: APSL コンパイラのコンパイルスピード

5.3 ローカル変数をインスタンス変数に変換した際に生じるオーバーヘッドの測定

`APSL` では，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できるようにするために，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している．そこで本節では，(a) インスタンス変数へのアクセス時間と (b) ローカル変数へのアクセス時間をそれぞれ測定した．測定方法として，クラスが `Object` で値が `null` の変数に 1,000,000 回 write アクセスを行いそこから 1 回アクセスを行う際の実行時間を得る，という作業を 5 回行いその平均を測定値とした．測定結果を表 5.2 に示す (単位 μs)．

(a)	(b)
0.35	0.30

表 5.2: ローカル変数をインスタンス変数に変換した際に生じるオーバーヘッド

結果より，インスタンス変数に変換した際に生じるオーバーヘッドは 16.67%と大きく，`main` メソッド内でローカル変数に大量にアクセスする場合は性能が低下してしまうことが分かった．しかしこれは現在の `APSL` コンパイラが `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しているためである．そのため，他のホストで用いる

ローカル変数のみをインスタンス変数に変換するようにすれば，ある程度性能低下を防ぐことができると考えられる．

5.4 コミュニケーションズパッケージを選択する時間

本節では，通信で使用できるかどうかを調べる変数の数を変化させてプレースエージェントがコミュニケーションズパッケージを選択する時間を測定した．測定方法として，通信で使用できるかどうか調べる変数の数に対するコミュニケーションズパッケージの選択にかかる実行時間を得る，という作業を通信で使用できるかどうかを調べる変数の数を変えてそれぞれ 10 回測定し，その平均を測定値とした．測定結果を図 5.2に示す．

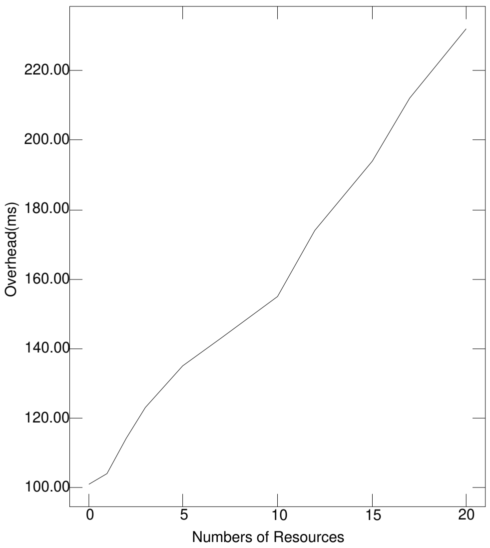


図 5.2: コミュニケーションズパッケージの選択にかかるオーバーヘッド

図 5.2より，調べる変数の数とコミュニケーションズパッケージを選択する時間との関係がほぼ比例し，規模拡張性があることが分かった．

5.5 変数へのアクセスを制御する際に生じるオーバーヘッドの測定

本節では，(a) アクセス制御している変数にソケットを用いてアクセスを行う際の実行時間と，(b) アクセス制御していない変数にソケットを用いてアクセスを行う際の実行時間をそれぞれ測定した．測定方法として，クラスが `Integer` で値が 0 の変数に 1,000

回アクセスを行いそこから 1 回アクセスを行う際の実行時間を得る，という作業を 5 回行いその平均を測定値とした．測定結果を表 5.3に示す (単位 ms)．

(a)	(b)
18.02	17.51

表 5.3: アクセスを制御する際にかかるオーバーヘッド

結果より，変数へのアクセスを制御する際に生じるオーバーヘッドは 2.95%であることが分かった．一般に他のエージェントが持つ変数へのアクセスはそれ程多くなく，またオーバーヘッドが 2.95%と小さい．そのため，変数のアクセス制御が APSL の性能を著しく低下させることはないと考えられる．

第 6 章

エージェントキャンパスガイドの試作

エージェントキャンパスガイドは，ユーザの位置情報と個人情報を元にして大学内の建物を案内する個人化を考慮した実世界ガイドシステムである．本章ではまず，実世界ガイドシステムの背景について説明し，続いて個人化を考慮した実世界ガイドシステム，エージェントガイドについて説明する．最後に，`APSL` を用いて実現したエージェントキャンパスガイドについて説明する．

6.1　背景

近年のコンピュータの小型化や無線ネットワーク技術の発展により，ユーザのいる場所に関わらずユーザの持つコンピュータをネットワークに接続することができるモーバイルコンピューティングが現実のものとなってきている．そして，人間の生活する現実世界とネットワーク上に実現されている仮想世界とをシームレスに連結し，ユーザの持つモーバイルコンピュータが位置情報を元にしてユーザを案内するシステム (本論文では，以後このシステムを実世界ガイドシステムと総称する) の研究が注目されている [1][22][24]．現在，実世界ガイドシステムとして，ウォークナビ [1]，ナビカム [22]，サイバーガイド [24] などが実装されている．しかしこれらのシステムは，システムを特定のユーザに適応する個人化を考慮しておらず，ユーザの興味や関心に基づいてユーザを案内することができない．そのため，例えば図書館内において，コンピュータ関連の本がどこにあるかということを案内することはできるが，ユーザの探している本がどこにあるかということを案内したり，ユーザが以前予約した本が届いたということを知らせたりすることはできない．

また，個人化を考慮したシステムとして，ポイントキャストといったユーザが選択した番組を各ユーザごとに提示するインターネット放送システムがある．個人化を考慮したシステムでは，ユーザに提示するコンテンツが各ユーザ毎に異なる．そして，コンテンツを構成する情報はネットワーク上に分散している．そこでインターネット放送システムでは，ユーザの要求が番組の選択という形で限定されているので，番組毎に集めた情報から作成したコンテンツを各ユーザに提示している．しかし，個人化を考慮した実世界ガイドシステムはインターネット放送システムとは異なり，ユーザの要求は多様多種である．例えば図書館内で，あるユーザは自分の研究に関連する本を探すことを要求するかもしれないし，またあるユーザは以前予約した本が届いているかどうかを確かめることを要求するかもしれない．そのため，個人化を考慮した実世界ガイドシステムをインターネット放送システムが用いた方法で実現すると，ユーザの要求を柔軟に満たすコンテンツを提示できなくなる．

そこで本論文では，ユーザの個人情報を持ってネットワーク上を動き回るモーバイルエージェントを用いて個人化を実現した実世界ガイドシステム，エージェントガイドを提案する．ユーザはモーバイルエージェントに個人情報を持たせてネットワークに移動させておけば，モーバイルエージェントがネットワーク上を動き回り，ネットワーク上に分散されている情報を集めて，ユーザの要求を満たすコンテンツを自動的に作成する．そのため，ユーザの要求を柔軟に満たすコンテンツを各ユーザに提示することができる．さらに本論文では，ユーザの位置情報や個人情報を元にして大学内の設備を案内するエージェントガイド，エージェントキャンパスガイドを試作する．

6.2 エージェントガイド

エージェントガイドは，ユーザの位置情報と個人情報を元にしてユーザが現在いる場所を案内する個人化を考慮した実世界ガイドシステムである．

6.2.1 ネットワーク環境

エージェントガイドのネットワーク環境を図 6.1に示す．ネットワーク環境には，カスタマーホスト，サービスホスト，エージェンシーホストという 3 種類のホストがそれぞれ存在する．

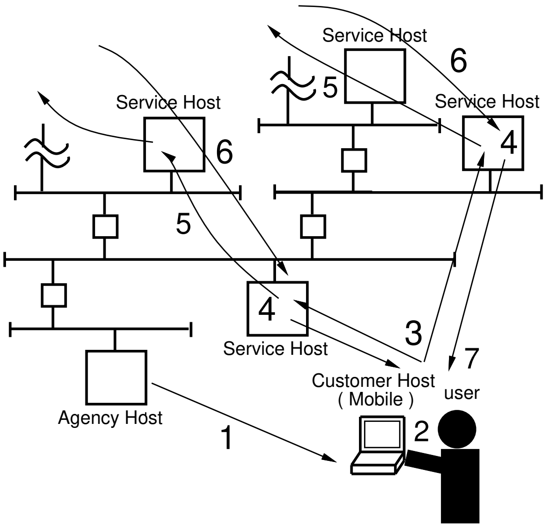


図 6.1: エージェントガイドのネットワーク環境

- カスタマーホスト

エージェントガイドを用いるユーザが手に持つモーバイルコンピュータ．カスタマーホストには，ネットワークに接続して通信を行う機器と赤外線信号を受信する機器がそれぞれ接続されている．またカスタマーホストには，ユーザの個人情報を管理して，サービスホストからサービスページを，エージェンシーホストからエージェンシーページをそれぞれダウンロードして表示するブラウザが常時起動されている．ブラウザはエージェンシーホストのホスト名は知っているが，サービスホストのホスト名は知らない．

- サービスホスト

  モーバイルエージェントに様々なサービスを提供して，場所を案内するサービスページをカスタマーホスト上のブラウザに提供するプレースエージェントが実行されているホスト．サービスホストは，場所ごとに1つづつ存在する．

- エージェンシーホスト

  ユーザの個人情報をブラウザに登録し，個人情報を持ったモーバイルエージェントを作成することができるエージェンシーページを提供するプレースエージェントが実行されているホスト．エージェンシーページには，場所の名前，場所に備え付けられている赤外線発信器が発信するID，場所に対応したサービスホストのホスト名をそれぞれ関連付けてるエージェンシーテーブルが付属している．そして，エージェンシーホストは，エージェントガイドのネットワーク環境下に1つしか存在しない．

6.2.2 実行の手順

エージェントガイドの実行手順を図6.1を用いて説明する．

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．するとエージェンシーホストからエージェンシーページが自動的にダウンロードされ，ブラウザ上に表示される．そしてブラウザは，エージェンシーページに付属しているエージェンシーテーブルを獲得する．
2. ユーザは，ユーザID，名前，これから行く場所，各場所で行いたいことといった個人情報を入力する．ユーザの入力が終了したら，個人情報がブラウザに保存され，ユーザがこれから行く各場所に対応したモーバイルエージェントがそれぞれ作成される．

3. ブラウザは，エージェンシーテーブルを用いて場所の名前から場所に対応したサービスホストのホスト名を獲得し，モーバイルエージェントをサービスホストに移動させる．この時点で，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ことができる．

4. モーバイルエージェントはユーザの個人情報を用いて，サービスホスト上で動作しているプレースエージェントと通信を行い，サービスページを作成する際に必要となる情報を収集する．

5. モーバイルエージェントは，もし今いるサービスホスト上のプレースエージェントから別のサービスホストを紹介されたら，そのホストに移動して情報を収集する．モーバイルエージェントはこの作業を繰り返すことによって，ネットワークに分散されている情報を収集する．

6. 情報を収集したモーバイルエージェントは，最初に訪れたサービスホストに戻る．そして，獲得した情報とユーザの個人情報からサービスページを作成して，サービスホストに登録する．

7. ユーザは，カスタマーホストをネットワークに接続して場所に移動する．すると，ブラウザは場所に備え付けられた赤外線発信器から ID を受信して，エージェンシーテーブルを用いて ID から場所に対応したサービスホストのホスト名を獲得する．そして，そのサービスホスト上のプレースエージェントから，6. でモーバイルエージェントが作成したサービスページをダウンロードして，表示させる．もしサービスホストにモーバイルエージェントが作成したサービスページがなければ，サービスホストがあらかじめ用意しているサービスページをダウンロードして，表示する．サービスページが表示されたら，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ことができる．ユーザは表示されたサービスページを操作して，要求を満たす．

8. 7. を繰り返す．

エージェントガイドでは，カスタマーホスト上で起動されているブラウザが予め知っているのは，エージェンシーホストのホスト名のみである．そしてブラウザは，サービスホストのホスト名や赤外線信号の ID などをエージェンシーホストが提供するエージェンシーページから獲得する．そのため，エージェントガイドで案内する場所を増やしたり，赤外線発信器の ID を変更したりする際には，エージェンシーページ内のエージェン

シーテーブルを修正するだけで良い．これより，エージェントガイドは規模拡張性があると考えられる．

6.3 エージェントキャンパスガイド

本論文では，`APSL`，Java[10][1]，Delphi[18][2]をそれぞれ用いてエージェントキャンパスガイドを試作した．

6.3.1 モーバイルエージェント

`APSL` では，モーバイルエージェントを実現するクラスとして，`_apsl_MobileAgent` クラスを用意している．そこで，ユーザの個人情報を持ってサービスホストを渡り歩き，各サービスホスト上のプレースエージェントから情報を収集するモーバイルエージェントを，`_apsl_MobileAgent` クラスを用いて実装した．

6.3.2 プレースエージェント

`APSL` では，プレースエージェントを実現するクラスとして，`_apsl_PlaceAgent` クラスを用意している．そこで，モーバイルエージェントに情報を提供し，カスタマーホスト上のブラウザにエージェンシーページやサービスページを提供するプレースエージェントを，`_apsl_PlaceAgent` クラスを用いて実装した．

6.3.3 サービスホスト，エージェンシーホスト

エージェンシーホストとサービスホストには，`4.2` で実装したプレースエージェントを配置させた．`APSL` は Java ベースであるため，エージェンシーホストやサービスホスト上には Java が動く OS が起動していなくてはならない．

6.3.4 カスタマーホスト

カスタマーホストには，Windows95 が動作する携帯型 PC/AT 互換機[3]を用いた (図 6.2)．

---

[1] JDK(Java Developer's Kit) 1.1.4 を使用

[2] Delphi 2 を使用

[3] 東芝 リブレット 50 を使用

図 6.2: カスタマーホスト

6.3.5 エージェンシーページ，サービスページ

エージェンシーページとサービスページは `java.awt.Panel` クラスを用いて実装した．

6.3.6 エージェンシーテーブル

エージェンシテーブルは `java.util.Vector` クラスを用いて実装し，建物の名前や赤外線信号の ID から建物に対応したサービスホストのホスト名を引けるようにした．

6.3.7 ブラウザ

ブラウザは Java を用いて実装し，サービスホストからサービスページを，エージェンシーホストからエージェンシーページをそれぞれダウンロードして，表示させるようにした．そして，Java のプログラムは赤外線信号を受信する機器を制御することができないので，Delphi を用いて赤外線信号を受信するプログラム `IR` を実装した．エージェンシーホストからエージェンシーページをダウンロードするプロトコルを以下に示す．

1. ブラウザは，エージェンシーホスト上で実行されているプレースエージェントにソケットの作成を要求する．ソケットが作成されると，ユーザ ID をプレースエージェントに送信する．
2. プレースエージェントは，ブラウザから送られて来たユーザ ID を受け取ると，引数としてユーザ ID を与えて `APSL` の `_apsl_PlaceAgent` クラスで提供されている `sendProgram` メソッドを実行させる．すると，エージェンシーページがブラウザに送信される．

3. ブラウザはエージェンシーページを受け取り，プレースエージェントとのソケットを閉じる．

続いて，赤外線信号を受信して，サービスホストからサービスページをダウンロードするプロトコルを以下に示す．

1. `IR` は赤外線信号を受信すると，ブラウザにソケットの作成を要求する．そしてソケットが作成されると，赤外線信号の ID を文字列としてブラウザに送信する．
2. ブラウザは `IR` から送られて来た ID を受け取ると，エージェンシーテーブルを用いて ID からサービスホストのホスト名を獲得する．そして `IR` とのソケットを閉じる．
3. ブラウザは，サービスホスト上で実行されているプレースエージェントにソケットの作成を要求する．ソケットが作成されると，ユーザ ID をプレースエージェントに送信する．
4. プレースエージェントは，ブラウザから送られて来たユーザ ID を受け取ると，引数としてユーザ ID を与えて `APSL` の `_apsl_PlaceAgent` クラスで提供されている `sendProgram` メソッドを実行する．すると，該当するサービスページがブラウザに送信される．
5. ブラウザはサービスページを受け取り，プレースエージェントとのソケットを閉じる．

6.3.8 実行例

本節では，中央図書館で Java に関する本を探し，購買部で 30 ドル以内のノートパソコンを探すユーザがエージェントキャンパスガイドを使用する例を説明する．

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．すると，エージェンシーホストからエージェンシーページが自動的にダウンロードされ，表示される．ユーザは表示されたページにユーザ ID，名前と中央図書館で Java に関する本を探し，購買部で 3000 ドル以内のノートパソコンを探すということを入力をする (図 6.3)．
2. ユーザは Regist ボタンを押す．すると，中央図書館で Java に関する本を探すモーバイルエージェント A と，購買部で 3000 ドル以内のノートパソコンを探すモーバイルエージェント B が作成される．そして，モーバイルエージェント A は中央図

User Registration Form

UserID: katsuno

Name: Yasuharu Katsuno

Please check all places you go:

☐ Cafeteria

☑ Central Library　List Java Related Books

☐ Computer Room

☐ Gymnasium

☑ University Shop　List Note PCs less than $3000

図 6.3: エージェンシーページ

書館に対応するサービスホスト A に，モーバイルエージェント B は購買部に対応するサービスホスト B にそれぞれ移動する．この時点で，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ことができる．

3. モーバイルエージェント A は，サービスホスト A 上のプレースエージェント A から中央図書館にある Java に関する本のリストを獲得する．ここで，モーバイルエージェント A は，プレースエージェント A から計算機科学図書館に対応するサービスホスト C を紹介されたので，サービスホスト C に移動する．そして，サービスホスト C 上のプレースエージェント C から，計算機科学図書館にある Java に関する本のリストを獲得する．最後に，サービスホスト A に戻って，獲得した本のリストを元にサービスページを作成して登録する．
4. モーバイルエージェント B は，サービスホスト B 上のプレースエージェント B から 3000 ドル以内の購買部で良く売れているノートパソコンのリストを獲得する．ここで，モーバイルエージェント B は，プレースエージェント B からリスト内の各ノートパソコンの性能や在庫，納期などを管理するサービスホスト D を紹介されたので，サービスホスト D に移動する．そして，サービスホスト D 上のプレースエージェント D から各ノートパソコンの性能や在庫，納期などのリストを獲得する．最後に，サービスホスト B に戻って，獲得したリストを元にサービスページを作成して登録する．
5. ユーザは中央図書館に移動し，カスタマーホストをネットワークに接続して中に入る．すると，ブラウザは中央図書館の入口に備えられた赤外線発信器から ID を

受信し (図 6.4)，サービスホスト A から 3. でモーバイルエージェントが作成したサービスページをダウンロードして表示させる (図 6.5)．この時点で，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ことができる．ユーザは借りたい本のラジオボタンをクリックし Reserve ボタンを押せば，本を借り出しを予約することができる．また，Another Library ボタンを押すと，計算機科学図書館内にある Java に関する本のリストが表示される (図 6.6)．ユーザは借りたい本のラジオボタンをクリックし Reserve ボタンを押せば，本を取り寄せて借り出しを予約することができる．

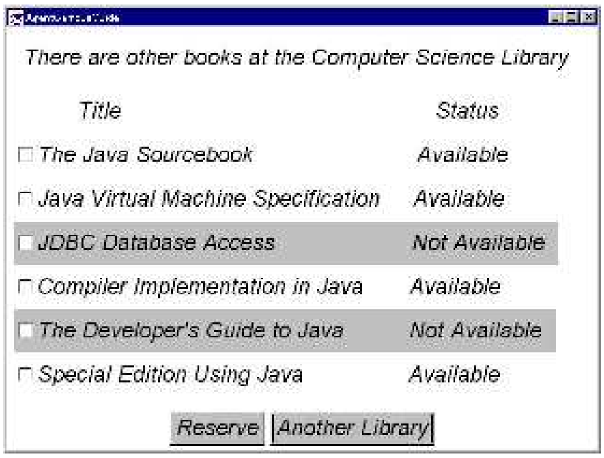
図 6.4: 赤外線発信器から ID を受け取る

Welcome to the Central Library, Yasuharu Katsuno

Here is a list of the results you requested.

Title	Status
☐ Advanced Java Programming	Available
☐ Developer Tools for Java	Not Available
☐ Introduction to Java	Available
☐ Java Developer Guide	Available
☐ Java Interactive Programming	Not Available

図 6.5: 中央図書館で獲得したページ (1/2)

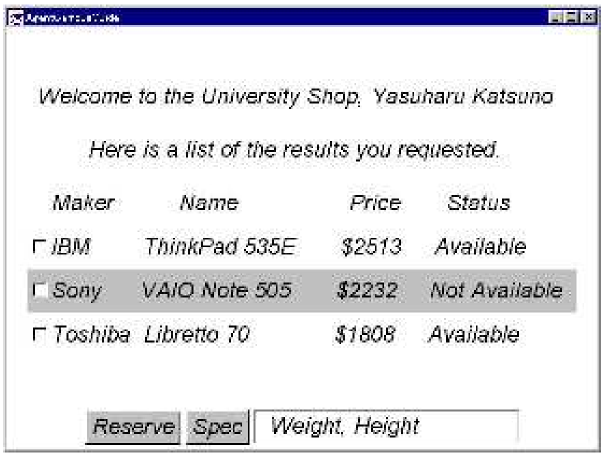


There are other books at the Computer Science Library

Title	Status
☐ The Java Sourcebook	Available
☐ Java Virtual Machine Specification	Available
☐ JDBC Database Access	Not Available
☐ Compiler Implementation in Java	Available
☐ The Developer's Guide to Java	Not Available
☐ Special Edition Using Java	Available

図 6.6: 中央図書館で獲得したページ (2/2)

6. ユーザは購買部に移動し，カスタマーホストをネットワークに接続して中に入る．すると，ブラウザは購買部の入口に備えられた赤外線発信器から ID を受信し，サービスホスト B から 4. でモーバイルエージェントが作成したサービスページをダウンロードして表示させる (図 6.7)．この時点で，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ことができる．ユーザは購入したいノートパソコンのラジオボタンをクリックし Reserve ボタンを押せば，ノートパソコンの購入を予約することができる．また性能や在庫数，納期などが知りたければ，キーワードを入力して Spec ボタンを押すと，各ノートパソコンの性能や在庫数，納期などが表示される (図 6.8)．



Welcome to the University Shop, Yasuharu Katsuno

Here is a list of the results you requested.

Maker	Name	Price	Status
☐ IBM	ThinkPad 535E	$2513	Available
☐ Sony	VAIO Note 505	$2232	Not Available
☐ Toshiba	Libretto 70	$1808	Available

Weight, Height

図 6.7: 購買部で獲得したページ (1/2)

Weight, Height

	Maker	Name	Weight	Height
☐	IBM	ThinkPad 535E	1.70kg	41.0mm
☐	Sony	VAIO Note 505	1.35kg	23.9mm
☐	Toshiba	Libretto 70	0.85kg	34.0mm

図 6.8: 購買部で獲得したページ (2/2)

第 7 章

今後の課題

今後の課題を以下に示す．

- 移動して通信を行なうか，今いるホストから通信を行なうかを自動的に選択する機構の導入

  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ためには，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，モーバイルエージェントのサイズによっては，移動して移動先のホスト上で通信を行うよりも，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て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自動的に選択する機構が有効である．この実現のために，通信コストを計算する機構をAPSLランタイムライブラリに導入することを検討している．

- コミュニケーションズパッケージを持つモーバイルエージェントの実現

  APSLでは，プレースエージェントのみがコミュニケーションズパッケージを持っている．もし，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を記述するプログラマが望む通信を行うことができる．この実現のために，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の記述構文の変更を検討している．

- サービスホスト上のリソースに対するアクセス制御の実現

  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，モーバイルエージェントがサービスホスト上のリソースを過剰に食い潰す恐れがある．そこで，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．これの実現のために，ホスト上のリソースへのアクセスを制御する機構をAPSLランタイムライブラリに導入することを検討している．

第 8 章

結論

モーバイルコンピュータをインターネットに接続しインターネット上で作業を行なう際には，ネットワーク上を動き回るモーバイルエージェントを用いたプログラミングが有効な手段であると考えられている．また，モーバイルエージェントの実現にJavaを用いるのが適していると考えられており，現在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，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Javaの構文の拡張を行い，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した．また，APSLのソースからJavaのソースに変換する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することなく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る．

本論文ではさらに性能評価を行ない，エージェントキャンパスガイドをアプリケーションとして実装することでAPSLの実用性を実証した．

今後の課題を以下にまとめる。

- 移動して通信を行なうか，今いるホストから通信を行なうかを自動的に選択する機構の導入
- コミュニケーションズパッケージを持つモーバイルエージェントの実現
- サービスホスト上のリソースに対するアクセス制御の実現

謝辞

本研究の機会を与えて下さり，また日頃から懇切なご指導を頂きました慶應義塾大学大学院理工学研究科計算機科学専攻客員教授 所 真理雄 博士に心から深く感謝致します．

また、本研究に対し、多大なる御指導と貴重な助言を下さった株式会社ソニーコンピュータサンエンス研究所 村田 賢一 氏に深く感謝致します．

そして，株式会社ソニーコンピュータサイエンス研究所 長尾 確 博士にはエージェントキャンパスガイドの試作の際に大変お世話になりました．長尾 博士に深く感謝致します．

さらに3年間苦楽を共に研究した修士2年生をはじめ、本研究室の皆様に深く感謝致します。

最後に、経済面や生活面などで影ながら研究生活を支援してくれた両親に深く感謝致します。

1998年2月

参考文献

[1] 長尾確，暦本純一，伊藤純一郎，早川由紀，八木正紀，安村通晃．ウォークナビ：ローケーションアウェアなインタラクティブ情報案内システム．インタラクティブシステムとソフトウェア *III*, 近代科学社，1995．

[2] Danny B.Lange and Daniel T. Chang. *IBM Aglets Workbench Programming Mobile Agents in Java A White Paper.* Technical Report, *IBM Corporation*, Sep 1996.

[3] Danny B.Lange and Daniel T. Chang. *IBM Aglets Workbench Programming Mobile Agents in Java A White Paper.* Technical Report, *IBM Corporation*, Sep 1996.

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

[5] IBM Corporation. *Mobile Agent Facility Specification.* Technical Report, *IBM Corporation*, 1996.

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

[7] H. Robert Frost. *Documentation for the Java(tm) Agent Template, Version 0.3. Stanford University*, 1995.

[8] Inc. General Magic. *An Introduction to Safety and Security in Telescript.* Technical Report, *General Magic,Inc.*, 1995.

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

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

[11] Robert Gray, David Kotz, Saurab Nog, Daniela Rus, and George Cybenko. *Mobile agents for mobile computing.* Technical Report, *Department of Computer Science, Dartmouth College*, May 1996.

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

[13] Robert S. Gray. *Agent Tcl: A transportable agent system.* Technical Report, *Department of Computer Science, Dartmouth College*, Nov 1995.

[14] Robert S. Gray. *Agent Tcl: Alpha Release 1.1.* Technical Report, *Department of Computer Science, Dartmouth College*, 1995.

[15] The Technology Partnership Group. *CyberAgent White Paper.* Technical Report, *The Technology Partnership Group*, 1996.

[16] 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.

[17] 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.

[18] C. Marco, G. Tim, and F. L. John. *Delphi Developer's Handbook. Sybex*, 1997.

[19] Sun Microsystems. *Java Object Serialization Specification.* Technical Report, Sun Microsystems, Inc., 1996.

[20] P.E.Clements, T.Papaioannou, and J.M.Edwards. Aglets: Enabling the Virtual Enterprise. *ME-SELA(Managing Enterprises - Stakeholders, Engineering, Logistics and Achievement)'97*, July 1997.

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

[22] Jun Rekimoto and Katashi Nagao. The World through the Computer: Computer Augmented Interaction with Real World Environments. *In Proceedings of UIST '95, ACM Symposium on User Interface Software and Technology*, 1995.

[23] Markus StraBer, Joachim Baumann, and Fritz Hohl. *MOLE - A Java Based Mobile Agent System.* Technical Report, *IPVR(Institude for Parallel and Distributed Computer Systems) University of Stuttgart*, May 1996.

[24] L. Sue, A. Dietmar, Gregory D. A., and Chris A. Cyberguide: Prototyping contextaware mobile applications. *In Proceedings of ACM CHI'96 Project Note*, 1996.

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

[26] James E. White. *Mobile Agents.* Technical Report, General Magic, Inc., Oct 1995.

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

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

Appendix A

APSLの記述構文

本章では，APSL の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の記述構文を LALR(1) 文法で示す．

Goal:
APSLDeclaration

APSLDeclaration:
$ImportDeclarations_{opt}$
APSLBodyDeclaration

$ImportDeclarations_{opt}$:
$ImportDeclarations_{opt}$ *ImportDeclaration*

ImportDeclaration:
`import` *Name* Mul_{opt} `;`

Mul_{opt}:
`. *`

APSLBodyDeclaration:
MobileAgentDeclaration
PlaceAgentDeclaration
CommPkgDeclaration

MobileAgentDeclaration:
`agent` *Id* { $MobileAgentBodyDeclarations_{opt}$ }

$MobileAgentBodyDeclarations_{opt}$:
$MobileAgentBodyDeclarations_{opt}$ *MobileAgentBodyDeclaration*

MobileAgentBodyDeclaration:
AccessControlFieldDeclaration `;`
MobileMethodDeclaration
MainMethodDeclaration

PlaceAgentDeclaration:
`agent` *Id* { $PlaceAgentBodyDeclarations_{opt}$ }

$PlaceAgentBodyDeclarations_{opt}$:
$PlaceAgentBodyDeclarations_{opt}$ *PlaceAgentBodyDeclaration*

PlaceAgentBodyDeclaration:
AccessControlFieldDeclaration `;`
PlaceMethodDeclaration
InitMethodDeclaration
CommMethodDeclaration

CommPkgDeclaration:
`communication` *Id* { *MessageTypeDeclaration ClientDeclaration ServerDeclaration* }

AccessControlFieldDeclaration:
Type $Access_{opt}$ *VariableName*
Type $Access_{opt}$ *VariableName* `=` *VariableInitializer*

$Access_{opt}$:
Permission

Permission:
`:[-]:`
`:[r]:`
`:[w]:`
`:[rw]:`

VariableName:
Id

VariableInitializer:
Expression
{ *VariableInitializers* }

VariableInitializers:
VariableInitializer
VariableInitializers , *VariableInitalizer*

MobileMethodDeclaration:
Modifier `void` *Id* ($ParameterList_{opt}$) *MobileMethodBody*
Modifier Type Id ($ParameterList_{opt}$) *MobileMethodBody*

PlaceMethodDeclaration:
Modifier `void` *Id* ($ParameterList_{opt}$) *PlaceMethodBody*
Modifier Type Id ($ParameterList_{opt}$) *PlaceMethodBody*

Modifier:
`nocomm`
`incomm`

$ParameterList_{opt}$
ParameterList

ParameterList:
ParameterDeclaration
ParameterList , *ParameterDeclaration*

ParameterDeclaration:
Type VariableName

MobileMethodBody:
MobileBlock
;

MainMethodDeclaration:
`main` ($ParameterList_{opt}$) *MainMethodBody*

MainMethodBody:
MainOutBlock

InitMethodDeclaration:
`init` ( ) *InitMethodBody*

InitMethodBody:
InitBlock

CommunicationMethodDeclaration:
`communication` ( ) *CommunicationMethodBody*

CommunicationMethodBody:
CommunicationBlock

PlaceMethodBody:
Block
;

MessageTypeDeclaration:
`MessageType` { $MessageTypeNames_{opt}$ }

$MessageTypeNames_{opt}$:
MessageTypeNames

MessageTypeNames:
MessageTypeName
MessageTypeNames , *MessageTypeName*

MessageTypeName:

Id
ClientDeclaration:
`client` (*ParameterList*$_{opt}$) { *StateBody*$_{opt}$ }
ServerDeclaration:
`server` (*ParameterList*$_{opt}$) { *StateBody*$_{opt}$ }
StateBody$_{opt}$*:*
StateBody
StateBody:
StateBodyStatement
StateBody StateBodyStatement
StateBodyStatement:
FieldDeclaration ;
Id CommPkgBlock
FieldDeclaration:
Type VariableName
Type VariableName = *VariableInitializer*
MobileBlock:
{ *MobileBlockBody*$_{opt}$ }
MobileBlockBody$_{opt}$
MobileBlockBody
MobileBlockBody:
MobileBlockBodyStatement
MobileBlockBody MobileBlockBodyStatement
MobileBlockBodyStatement:
LocalVariableDeclaration ;
MobileStatement
LocalVariableDeclaration:
VariableDeclarator
VariableDeclarator:
Type VariableName
Type VariableName = *VariableInitializer*
MobileStatement:
MobileBlock
EmptyStatement
ExpressionStatement
MobileSwitchStatement
MobileDoStatement
BreakStatement
ContinueStatement
ReturnStatement
ThrowStatement
MobileTryStatement
MobileLabelStatement
MobileIfStatement
MobileWhileStatement
MobileForStatement
MobileCommunicationStatement
MobileCommunicationsStatement:

`communicate` `(` *ServiceName* `)` { *ResourceNames*$_{opt}$ } `reply` { *ResourceNames*$_{opt}$ }
ServiceName:
Id
ResourceNames$_{opt}$*:*
ResourceNames
ResourceNames:
ResourceName
ResourceNames `,` *ResourceName*
ResourceName:
Access$_{opt}$ *VariableName* *DimensionExpressions*$_{opt}$
EmptyStatement:
`;`
MobileLabelStatement:
Id `:` *MobileAgentStatement*
MobileIfStatement:
`if` `(` *Expression* `)` *MobileBlock* *MobileElseStatement*$_{opt}$
MobileElseStatement$_{opt}$*:*
`else` *MobileBlock*
`else if` `(` *Expression* `)` *MobileBlock* *MobileElseStatement*$_{opt}$
MobileSwitchStatement:
`switch` `(` *Expression* `)` *MobileSwitchBlock*
MobileSwitchBlock:
{ *MobileCaseBlockParts* *SwitchLabel*$_{opt}$ }
{ *SwitchLabel*$_{opt}$ }
MobileCaseBlockParts:
MobileCaseBlockPart
MobileCaseBlockParts *MobileCaseBlockPart*
MobileCaseBlockPart:
SwitchLables *MobileBlockBody*
SwitchLabels:
SwitchLabel
SwitchLabels *SwitchLabel*
SwitchLabel$_{opt}$*:*
SwitchLabel
SwitchLabel:
`case` *ConstanceExpression* `:`
`default :`
MobileWhileStatement:
`while` `(` *Expression* `)` *MobileBlock*
MobileDoStatement:
`do` *Mobileblock* `while` `(` *Expression* `)` `;`
MobileForStatement:
`for` `(` *ForInitializer*$_{opt}$ `;` *Expression*$_{opt}$ `;` *ForUpdate*$_{opt}$ `)` *MobileBlock*
`for` `(` *LocalVariableDeclaration* `;` *Expression*$_{opt}$ `;` *ForUpdate*$_{opt}$ `)` *MobileBlock*
ForInitializer$_{opt}$
StatementExpressionList
ForUpdate$_{opt}$*:*
StatementExpressionList

StatementExpressionList:
  StatementExpression
  StatementExpressionList `,` *StatementExpression*
BreakStatement:
  `break` *Id*$_{opt}$ `;`
ContinueStatement:
  `continue` *Id*$_{opt}$ `;`
ReturnStatement:
  `return` *Expression*$_{opt}$ `;`
ThrowStatement:
  `throw` *Expression* `;`
MobileTryStatement:
  `try` *MobileBlock MobileCatches*
  `try` *MobileBlock* $MobileCatches_{opt}$ *MobileFinally*
$MobileCatches_{opt}$*:*
  MobileCatches
MobileCatches:
  MobileCatch
  MobileCatches MobileCatch
MobileCatch:
  `catch` `(` *ParameterDeclaration* `)` *MobileBlock*
MobileFinally:
  `finally` *MobileBlock*
MainOutBlock:
  { $MainOutBlockBody_{opt}$ }
$MainOutBlockBody_{opt}$*:*
  MainOutBlockBody
MainOutBlockBody:
  MainOutBlockBodyStatement
  MainOutBlockBody MainOutBlockBodyStatement
MainOutBlockBodyStatement:
  MainLocalVariableDeclaration `;`
  MainOutStatement
MainLocalVariableDeclaration:
  VariableDeclarator
MainOutStatement:
  MainOutBlock
  EmptyStatement
  DoOutStatement
  IfOutStatement
  WhileOutStatement
  ForOutStatement
  MoveStatement
MoveStatement:
  `host` `(` *HostName* `)` *MainInBlock*
HostName:
  Expression
IfOutStatement:

`if ( ` *Expression* `)` *MainOutBlock* *ElseOutStatement*$_{opt}$
ElseOutStatement$_{opt}$
`else` *MainOutBlock*
`else if (` *Expression* `)` *MainOutBlock* *ElseOutStatement*$_{opt}$
WhileOutStatement:
`while (` *Expression* `)` *MainOutBlock*
DoOutStatement:
`do` *MainOutBlock* `while (` *Expression* `) ;`
ForOutStatement:
`for (` *ForInitializer*$_{opt}$ `;` *Expression*$_{opt}$ `;` *ForUpdate*$_{opt}$ `)` *MainOutBlock*
`for (` *MainLocalVariableDeclaration* `;` *Expression*$_{opt}$ `;` *ForUpdate*$_{opt}$ `)` *MainOutBlock*
MainInBlock:
{ *MainInBlockBody*$_{opt}$ }
MainInBlockBody$_{opt}$*:*
MainInBlockBody
MainInBlockBody:
MainInBlockBodyStatement
MainInBlockBody *MainInBlockBodyStatement*
MainInBlockBodyStatement:
MainLocalVariableDeclaration `;`
MainInStatement
MainInStatement:
MobileStatement
InitBlock:
{ *InitBlockBody*$_{opt}$ }
InitBlockBody$_{opt}$*:*
InitBlockBody
InitBlockBody:
InitBlockBodyStatement
InitBlockBody *InitBlockBodyStatement*
InitBlockBodyStatement:
`registercommunication (` *RegistName* `,` *CommPkgName* `) ;`
Statement
RegistName:
Expression
CommPkgName:
Id
CommBlock:
{ *CommBlockBody*$_{opt}$ }
CommBlockBody$_{opt}$*:*
CommBlockBody
CommBlockBody:
CommBlockBodyStatement
CommBlockBody *CommBlockBodyStatement*
CommBlockBodyStatement:
`service (` *ServiceName* `)` { *ServiceBlockBodyStatement*$_{opt}$ }
ServiceBlockBodyStatement$_{opt}$*:*
ServiceBlockBody

ServiceBlockBody:
- *ServiceBlockBodyStatement*
- *ServiceBlockBody ServiceBlockBodyStatement*

ServiceBlockBodyStatement:
- `case [` *CheckedNames*$_{opt}$ `] : start (` *RegistName* `) ;`
- `default : comm_error ( ) ;`

CheckedNames$_{opt}$*:*
- *CheckedNames*

CheckedNames:
- *CheckedName*
- *CheckedNames* **,** *CheckedName*

CheckedName:
- *Id*

PlaceBlock:
- { *PlaceBlockBody*$_{opt}$ }

PlaceBlockBody$_{opt}$
- *PlaceBlockBody*

PlaceBlockBody:
- *PlaceBlockBodyStatement*
- *PlaceBlockBody PlaceBlockBodyStatement*

PlaceBlockBodyStatement:
- *LocalVariableDeclaration* ;
- *PlaceStatement*

PlaceStatement:
- *PlaceBlock*
- *EmptyStatement*
- *ExpressionStatement*
- *PlaceSwitchStatement*
- *PlaceDoStatement*
- *BreakStatement*
- *ContinueStatement*
- *ReturnStatement*
- *ThrowStatement*
- *PlaceTryStatement*
- *PlaceLabelStatement*
- *PlaceIfStatement*
- *PlaceWhileStatement*
- *PlaceForStatement*

PlaceLabelStatement:
- *Id* : *PlaceStatement*

PlaceIfStatement:
- `if (` *Expression* `)` *PlaceBlock PlaceElseStatement*$_{opt}$

PlaceElseStatement$_{opt}$*:*
- `else` *PlaceBlock*
- `else if (` *Expression* `)` *PlaceBlock PlaceElseStatement*$_{opt}$

PlaceSwitchStatement:
- `switch (` *Expression* `)` *PlaceSwitchBlock*

PlaceSwitchBlock:

{ *PlaceCaseBlockParts* *PlaceSwitchLabel*$_{opt}$ }
{ *PlaceSwitchLabel*$_{opt}$ }
PlaceCaseBlockParts:
PlaceCaseBlockPart
PlaceCaseBlockParts *PlaceCaseBlockPart*
PlaceCaseBlockPart:
SwitchLables *PlaceBlockBody*
PlaceWhileStatement:
`while` `(` *Expression* `)` *PlaceBlock*
PlaceDoStatement:
`do` *Placeblock* `while` `(` *Expression* `)` `;`
PlaceForStatement:
`for` `(` *ForInitializer*$_{opt}$ `;` *Expression*$_{opt}$ `;` *ForUpdate*$_{opt}$ `)` *PlaceBlock*
`for` `(` *LocalVariableDeclaration* `;` *Expression*$_{opt}$ `;` *ForUpdate*$_{opt}$ `)` *PlaceBlock*
PlaceTryStatement:
`try` *PlaceBlock* *PlaceCatches*
`try` *PlaceBlock* *PlaceCatches*$_{opt}$ *PlaceFinally*
PlaceCatches$_{opt}$:
PlaceCatches
PlaceCatches:
PlaceCatch
PlaceCatches *PlaceCatch*
PlaceCatch:
`catch` `(` *ParameterDeclaration* `)` *PlaceBlock*
PlaceFinally:
`finally` *PlaceBlock*
CommPkgBlock:
{ *CommPkgBlockBody*$_{opt}$ }
CommPkgBlockBody$_{opt}$:
CommPkgBlockBody
CommPkgBlockBody:
CommPkgBlockBodyStatement
CommPkgBlockBody *CommPkgBlockBodyStatement*
CommPkgBlockBodyStatement:
LocalVariableDeclaration `;`
CommPkgStatement
CommPkgStatement:
CommPkgBlock
EmptyStatement
ExpressionStatement
CommPkgSwitchStatement
CommPkgDoStatement
BreakStatement
ContinueStatement
CommPkgTryStatement
CommPkgLabelStatement
CommPkgIfStatement
CommPkgWhileStatement

CommPkgForStatement
NextStatement
SendStatement
ReceiveStatement
CommPkgLabelStatement:
Id `:` *CommPkgStatement*
CommPkgIfStatement:
`if` `(` *Expression* `)` *CommPkgBlock* *CommPkgElseStatement*$_{opt}$
CommPkgElseStatement$_{opt}$*:*
`else` *CommPkgBlock*
`else` `if` `(` *Expression* `)` *CommPkgBlock* *CommPkgElseStatement*$_{opt}$
CommPkgSwitchStatement:
`switch` `(` *Expression* `)` *CommPkgSwitchBlock*
CommPkgSwitchBlock:
{ *CommPkgCaseBlockParts* *SwitchLabel*$_{opt}$ }
{ *SwitchLabel*$_{opt}$ }
CommPkgCaseBlockParts:
CommPkgCaseBlockPart
CommPkgCaseBlockParts *CommPkgCaseBlockPart*
CommPkgCaseBlockPart:
SwitchLables *CommPkgBlockBody*
CommPkgWhileStatement:
`while` `(` *Expression* `)` *CommPkgBlock*
CommPkgDoStatement:
`do` *CommPkgBlock* `while` `(` *Expression* `)` `;`
CommPkgForStatement:
`for` `(` *ForInitializer*$_{opt}$ `;` *Expression*$_{opt}$ `;` *ForUpdate*$_{opt}$ `)` *CommPkgBlock*
`for` `(` *LocalVariableDeclaration* `;` *Expression*$_{opt}$ `;` *ForUpdate*$_{opt}$ `)` *CommPkgBlock*
CommPkgTryStatement:
`try` *CommPkgBlock* *CommPkgCatches*
`try` *CommPkgBlock* *CommPkgCatches*$_{opt}$ *CommPkgFinally*
CommPkgCatches$_{opt}$*:*
CommPkgCatches
CommPkgCatches:
CommPkgCatch
CommPkgCatches *CommPkgCatch*
CommPkgCatch:
`catch` `(` *ParameterDeclaration* `)` *CommPkgBlock*
CommPkgFinally:
`finally` *CommPkgBlock*
NextStatement:
`next` `(` *Id* `)` `;`
SendStatement:
`send` `(` *MessageTypeName* `,` *Expression* `)` `;`
ReceiveStatement:
`receive` `(` *Id* `)` *CommPkgSwitchBlock*
ExpressionStatement:
StatementExpression `;`

```
StatementExpression:
    AssignmentExpression
    PreIncrementExpression
    PreDecrementExpression
    PostIncrementExpression
    MethodInvocation
    ClassInstanceCreationExpression
Primary:
    PrimaryNotArrayCreation
    ArrayCreationExpression
PrimaryNotArrayCreation:
    Literal
    this
    ( Expression )
    ClassInstanceCreationExpression
    ArrayAccess
    MethodInvocation
    FieldAccess
ArrayCreationExpression:
    new ClassType DimesionExpressions
DimensionExpressions:
    DimensionExpression
    DimensionExpressions DimensionExpression
DimensionExpressions_opt
    DimensionExpression
DimensionExpression:
    [ Expression ]
ClassInstanceCreationExpression:
    new ClassType ( Arguments_opt )
Arguments_opt
    Arguments
Arguments:
    Expression
    Arguments , Expression
FieldAccess:
    super . Id
    Primary . Id
MethodInvocation:
    Name ( Arguments_opt )
ArrayAccess:
    Id [ Expression ]
    PrimaryNotArrayCreation [ Expression ]
UnaryExpression:
    PreIncrementExpression
    PreDecrementExpression
    UnaryExpressionNotPlusMinus
PreIncrementExpression:
    ++ UnaryExpression
```

```
PreDecrementExpression:
    -- UnaryExpression
UnaryExpressionNotPlusMinus:
    PostFixExpression
    ! UnaryExpression
    CastExpression
CastExpression:
    ( Expression ) UnaryExpressionNotPlusMinus
PostFixExpression:
    Primary
    PostIncrementExpression
    PostDecrementExpression
    Name
PostIncrementExpression:
    PostFixExpression ++
PostDecrementExpression:
    PostFixExpression --
MulticativeExpression:
    UnaryExpression
    MulticativeExpression * UnaryExpression
    MulticativeExpression / UnaryExpression
    MulticativeExpression % UnaryExpression
AdditiveExpression:
    MulticativeExpression
    AdditiveExpression + MulticativeExpression
    AdditiveExpression - MulticativeExpression
ShiftEpxression:
    additiveExpression
    ShiftExpression << AdditiveExpression
    ShiftExpression >> AdditiveExpression
    ShiftExpression >>> AdditiveExpression
RelationalExpression:
    ShiftExpression
    RelationnalExpression > ShiftExpression
    RelationnalExpression < ShiftExpression
    RelationnalExpression <= ShiftExpression
    RelationnalExpression >= ShiftExpression
    RelationnalExpression instanceof Type
EqualityExpression:
    RelationalExpression
    EqualityExpression == RelationalExpression
    EqualityExpression != RelationalExpression
AndExpression:
    EqualityExpression
    AndExpression & EqualityExpression
ExclusiveOrExpression:
    AndExpression
    ExclusiveOrExpression ^ AndExpression
```

InclusiveOrExpression:
    ExclusiveOrExpression
    InclusivOrExpression `|` *ExclusiveOrExpression*
ConditionalAndExpression:
    InclusiveOrExpression
    ConditionalAndExpression `&&` *InclusiveOrExpression*
ConditionalOrExpression:
    ConditionalAndExpression
    ConditionalOrExpression `||` *ConditionalAndExpression*
ConditionalExpression:
    ConditionalOrExpression
    ConditionalOrExpression `?` *Expression*
AssignmentExpression:
    LeftHandExpression AssignmentOperator AssignmentTarget
AssignmentTarget:
    ConditionalExpression
    AssignmentExpression
LeftHandExpression:
    Name
    FieldAccess
    ArrayAccess
AssignmentOperator:
    `=`
    `*=`
    `%=`
    `+=`
    `-=`
    `<<=`
    `>>=`
    `>>>=`
    `&=`
    `^=`
    `|=`
Expression$_{opt}$*:*
    Expression
Expression:
    AssignmentExpression
    ConditionalExpression
ConstanceExpression:
    Expression
Id$_{opt}$*:*
    Id
Id:
    Symbol
Type:
    ClassType
    ArrayType
ClassType:

Id
ArrayType:
ArrayType `[ ]`
Id `[ ]`
Name:
SimpleName
QualifiedName
SimpleName:
Id
QualifiedName:
Name `.` *SimpleName*
Literal:
IntegerLiteral
FloatingPointLiteral
CharacterLiteral
StringLiteral
BooleanLiteral
NullLiteral