契約番号 98 第 021 号

高度情報化支援ソフトウェアシーズ育成事業

高階モーバイルエージェントシステムの設計と実装

プログラムの説明書

平成11年2月

（株）インパクト

第1章　はじめに

通信ネットワークの普及により、プログラムをネットワークを通して配布することが可能になっている。しかし、それにはプログラムの明示的なダウンロード操作が必要であり、さらに、そのプログラムの起動時や既存プログラムの更新時にはシステム処理の中断が必要となるという問題がある。この研究では、モーバイルエージェントの枠組みを発展・拡張させたシステムを設計・実装しながらこれらの問題を解決していく。このシステムは、従来のエージェントの移動性・協調に加えて、モーバイルエージェント同士の自律的な合成や動的置換・更新などの既存システムにおいては困難であった機能を実現するものである。このシステムにより、ソフトウェアのオンラインバージョンアップや、処理内容や計算規模の変化に対する適応性をもつシステム実現し、進化的分散計算システムの実現に道を開く。

そこで、この研究では次世代のモーバイルエージェントシステムを設計・実装する。このシステムは、エージェントの自律的な移動に加えて、エージェントの動的な合成・置換を可能にする。具体的には、モーバイルエージェント内部に複数のモーバイルエージェントを含むことのできる高階モーバイルエージェントを導入し、エージェントの移動先としてコンピュータだけでなく、他のエージェントもその対象とすることによって、エージェントの移動を通じてエージェント集合体の構造変化を可能にする。また、さらにエージェント同士の動的な置換機構を提供し、実行中エージェントを別のエージェントで置換・拡張できるようにする。エージェント間の結合・協調関係を記述するルール型スクリプティング言語を導入し、柔軟かつ適応性をもったエージェント合成を可能にする。

1.1　研究開発の内容と構成

この研究ではモーバイルエージェントの枠組みを通じて行う。分散計算システムの自律的なオンラインバージョンアップや動的適応・拡張性を実現するには、システム構成要素となるエージェントの自立的な移動に加え、ネットワークを通して到着したエージェントを、システム構成要素の一つとしてシステムに自動合成することや、到着したエージェントによりシステム中のエージェントを動的置換・拡張することが必要となる。

しかし、既存のモーバイルエージェントシステムに克服すべき問題がある。例えば、既存システムは、移動や計算をエージェントという粒度の小さいプログラムを単位としているが、その一方でエージェント間の合成手法は欠如し、また、エージェント間の協調動作も非同期メッセージ通信などの単純かつ特定の疎結合に限定されている。さらに、

モーバイルエージェント同士の動的な置換や機能拡張は考慮されていない。

この研究では以下の三つの概念をもつモーバイルエージェントシステムを設計・実装しながら上記の問題点を解決する。

(1) 高階モーバイルエージェント、

(2) モーバイルエージェントの置換機構、

(3) モーバイルエージェントのための協調記述言語

モーバイルエージェントのもっとも基本な計算メカニズムはエージェントの移動である。そこで、エージェントの動的な合成をエージェントの移動として実現する。つまり、(1) はモーバイルエージェントの移動先として、コンピュータだけでなく、別のモーバイルエージェントに対しても移動可能にすると同時に、一つのモーバイルエージェントの内部に複数のモーバイルエージェントが入れ子状に存在できるようにする。そして、エージェントの移動を通じて、エージェント合成体の構造変更と拡張を実現する。なお、既存システムには、プレースと呼ばれる概念により複数のモーバイルエージェントを含有可能な計算実体を提供するものがあるが、プレース自体に移動能力はない。

次に (2) について述べる。実行中エージェントを別のエージェントで置き換えるためには、エージェントがもつ実行状態及びエージェント間協調関係を継承する必要があり、置換前にエージェント間の整合性を判定する必要がある。(2) では、エージェントを構成するプログラム (Java 言語のプログラム) のクラス名、インスタンス変数名、メソッド名、データ型などの情報（を符号化した 64 ビットハッシュ）が一致する場合はエージェントのインスタンス変数（メソッド変数は含めない）を継承しながら置換し、外部インタフェースのみ整合性をとれるときはエージェントの協調関係のみを継承する。

(3) モーバイルエージェントの集合体は、構成要素のエージェントや集合全体の移動を通じてその構成や計算環境が変化する。このため、オブジェクト指向計算でみられるような静的な関係でモーバイルエージェントを結びつけることはできない。そこで、エージェントの合成・協調方法を規定するスクリプティング言語を導入する。既存のエージェント記述言語は、個々のエージェントの通信・動作をターゲットとしているのに対し、この研究では、個々のエージェントの動作記述はエージェントプログラムで与えることとし、このスクリプティング言語は、エージェント間の結合関係や通信、リソース情報などの各種サービスをメタ定義するものとなる。これにより例えば、到着したエージェントを内包可能性、エージェント間の協調手順、移動先の計算機環境に合致したシステム構成変更を実現する。

このモーバイルエージェントシステム及びスクリプティング言語の設計では、その意味論を並列計算理論であるプロセス代数、その中でも R.Milner の π 計算や L.Cardelli の Mobile Ambients を設計背景におく。これにより、その理論的な特性を明らかにするとともに検証可能性を与える。

なお、この研究で開発するソフトウェアは、 高階モーバイルエージェント及びエージェントの動的合成・置換を実現するためのランタイムシステム、 協調スクリプティング言語処理系、 モーバイルエージェントの作成のためのフレームワークライブラリ

から構成される。なお、実装は Java 言語を用い、その実行も同言語の仮想機械上で実行する予定である。これによりハードウェアや OS に非依存なものとして構築される。

1.2 本報告書の構成

ここで以降の章構成を示す。

- 第 2 章 プログラムの構成

  プログラムの構成 (クラス階層)

- 第 3 章 プログラムの機能

  高階モーバイルエージェントの実現機能

  モーバイルエージェントの動的置換機能

  協調スクリプティング言語処理系の機能

  フレームワークライブラリの機能

  MobileSpaces システムの実装

- 第 4 章 プログラムの稼働環境

  ハードウェア

  オペレーティングシステム

  支援ソフトウェア

- 第 5 章 プログラムの操作

  MobileSpaces システムのインストール方法

  MobileSpaces システムの起動方法

  MobileSpaces システムの操作方法

  MobileSpaces のサンプルエージェント

第2章　プログラムの構成

本章では MobileSpaces のシステム構成を説明する。

2.1　システム構成

プログラムは、高階モーバイルエージェントを実現するランタイムシステムと、モーバイルエージェントを記述するためのフレームワークライブラリからなる。

高階モーバイルエージェントのランタイムシステム
: モーバイルエージェントの実行制御、移動、永続化などの各種機能を実現すると同時に、モーバイルエージェントの階層化を実現する。つまり、モーバイルエージェントの移動先として、コンピュータだけでなく、別のモーバイルエージェントに対しても移動可能にすると同時に、一つのモーバイルエージェントの内部に複数のモーバイルエージェントが入れ子状に存在できるようにする。そして、エージェントの移動を通じて、エージェント合成体の構造変更と拡張を実現する。

協調スクリプティング言語処理系
: モーバイルエージェントの集合体は、構成要素のエージェントや集合全体の移動を通じてその構成や計算環境が変化する。このため、オブジェクト指向計算でみられるような静的な関係でモーバイルエージェントを結びつけることはできない。そこで、エージェントの合成・協調方法を規定するスクリプティング言語を導入する。

高階モーバイルエージェントのフレームワークライブラリ
: モーバイルエージェントによる計算で不可欠となるエージェント間通信や各種リソースの取得に関する API 群からなり、さらにエージェント階層を取り扱うための API 群も同時に提供する

これらはすべて Java 言語のクラスファイルと提供される。これのクラス構成は以下のようにになる。

```
mobilespaces -|
              |- event -|
              |         |- AmbientEvent.class
              |         |- AmbientEventDispatcher.class
              |         |- AmbientEventListener.class
```

```
|          |- AmbientEventQueue.class
|          |- AmbientEventQueueItem.class
|          |- AmbientQueue.class
|          |- StatusEvent.class
|          |- StatusListener.class
|          |- StructureEvent.class
|          |- StructureListener.class
|
|- monitor -|
|            |- AmbientActionControl.class
|            |- AmbientButton.class
|            |- AmbientDestDialog.class
|            |- AmbientDialog.class
|            |- AmbientMenu.class
|            |- AmbientMonitor.class
|            |- AmbientTableControl.class
|            |- AmbientTableEvent.class
|            |- AmbientTableListener.class
|            |- AmbientTableMap.class
|            |- AmbientTableModel.class
|            |- AmbientTableSorter$1.class
|            |- AmbientTableSorter.class
|            |- AmbientTableView.class
|            |- AmbientTreeControl.class
|            |- AmbientTreeEvent.class
|            |- AmbientTreeListener.class
|            |- AmbientTreeNode.class
|            |- AmbientTreeView.class
|            |- AmbientWindowClosing.class
|
|- protocol -|
|             |- http -|
|             |        |- AmbientHTTPContent.class
|             |        |- AmbientHTTPContent.class
|             |        |- AmbientHTTPRequest.class
|             |        |- AmbientHTTPResponse.class
|             |
|             |- matp -|
|             |        |- AmbientMATPFactory.class
|             |        |- AmbientMATPHandler.class
```

```
|            |        |- AmbientMATPRequest.class
|            |
|            |- AmbientRequest.class
|            |- AmbientRequestFactory.class
|            |- AmbientRequestParser.class
|            |- AmbientResponse.class
|            |- AmbientResponseFactory.class
|            |- AmbientResponseParser.class
|
|- system -|
|          |- Ambient.class
|          |- AmbientByteCode.class
|          |- AmbientCallMessage.class
|          |- AmbientCallback.class
|          |- AmbientCallbackManager.class
|          |- AmbientClassLoader.class
|          |- AmbientCodeBase.class
|          |- AmbientConnection.class
|          |- AmbientContainer.class
|          |- AmbientContextImpl.class
|          |- AmbientControl.class
|          |- AmbientControlImpl.class
|          |- AmbientDefaultDispatcher.class
|          |- AmbientEvent.class
|          |- AmbientEventDispatcher.class
|          |- AmbientFrame.class
|          |- AmbientFrameWrapper.class
|          |- AmbientFuture.class
|          |- AmbientIdentifier.class
|          |- AmbientInfo.class
|          |- AmbientInternalFrame.class
|          |- AmbientJFrame.class
|          |- AmbientListener.class
|          |- AmbientLoader.class
|          |- AmbientLoaderStub.class
|          |- AmbientManager.class
|          |- AmbientManagerImpl.class
|          |- AmbientMessage.class
|          |- AmbientMessageManager.class
|          |- AmbientMessageQueue.class
```

```
|          |- AmbientMethod.class
|          |- AmbientNameManager.class
|          |- AmbientObject.class
|          |- AmbientPanel.class
|          |- AmbientQueue.class
|          |- AmbientRecord.class
|          |- AmbientRuntime.class
|          |- AmbientRuntimeImpl.class
|          |- AmbientServer.class
|          |- AmbientStatus.class
|          |- AmbientThread.class
|          |- AmbientThreadGroup.class
|          |- AmbientTimeout.class
|          |- AmbientURL.class
|          |- AmbientURLConnection.class
|
|- zip -|
|       |- AmbientJar.class
|       |- AmbientManifest.class
|       |- AmbientManifestField.class
|       |- AmbientReadJar.class
|       |- AmbientReadManifest.class
|       |- AmbientWriteManifest.class
|       |- AmbientWriteJar.class
|       |- AmbientZipOutputStream.class
|       |- AmbientZipUtil.class
|
|- Context.class
|- Future.class
|- Message.class
|- NoSuchAmbientException.class
|- NoSuchCallbackException.class
|- NoSuchLocationException.class
|- NoSuchServiceException.class
|- Station.class
|- TimeoutException.class
```

2.2 高階モーバイルエージェントのランタイムシステム

ランタイムシステムは Java 言語のアプリケーションとして構築されたプログラムであり、下記のような構成をとる。

ランタイムマネジャ
ランタイムシステムそのものの起動及び初期化を行い。また、終了時には所定の処理を実行する。

ランタイムイニシャライザ
通信設定を中心にランタイムシステムの初期化を行う。

ランタイムカーネル
ランタイムシステムの各種制御を提供する部分である。

ブート アップローダ
ランタイムシステム起動時に所定のエージェントをロード・実行開始する。ロード及び実行開始には、エージェントローダとエージェントマネジャーを利用する。

関連クラス：

- `AmbientManager.class`,
- `AmbientManagerImpl.class`,
- `AmbientNameManager.class`,
- `AmbientRuntime.class`,
- `AmbientRuntimeImpl.class`

エージェント移動マネジャ
エージェントのコンピュータ間移動を実現する。

エージェントセンダ
直列化マネジャーによりデータ列かされたエージェントを他のコンピュータに移動させる部分。TCP のクライアントとして構成される。

エージェントレシーバ
他のコンピュータから移動してくるエージェントをまつ。TCP のサーバとして構成され、受信後直ちにエージェントかメッセージを判別する。

関連クラス：

- AmbientConnection.class,
- AmbientServer.class,
- AmbientSender.class,
- AmbientReceiver.class

エージェント永続化マネジャ

エージェントをファイルを含むビットストリームに変換・逆変換する。

エージェントライター

エージェントをビットストリームに変換

エージェントリーダ

ビットストリームをエージェントに変換

ファイルマネジャ

ビットストリームの読み書きを行う

関連クラス：

- AmbientByteCode.class,
- AmbientLoader.class,
- AmbientLoaderStub.class,
- AmbientObject.class

エージェントローダマネジャ

エージェントのローディングを行う。なお、ローディングはエージェントの初期化した後に実行開始する方法と、実行状態状態を保持しているエージェントについてはその実効状態から処理を再開・継続することができる。

エージェントローダ

エージェントから、コード（Java 言語のクラス) と実効状態、及び各種属性情報を抽出する。

エージェントシリアライザ

バイト列化されたエージェントの実効状態を再びメモリ上のイメージに変換及びその逆変換をする。

クラスローダ

エージェントの実行に必要なファイルをロードする。

関連クラス：

- AmbientByteCode.class,
- AmbientLoader.class,
- AmbientLoaderStub.class,
- AmbientObject.class

エージェント 制御マネジャー

各エージェントの実効状態（初期化、実行、永続化、停止）を制御・実現する。

エージェント 実行制御マネジャー

エージェントの状態制御を行う。

エージェント スレッド 制御マネジャー

エージェントの能動実行を実現する。能動性はJava言語のjava.lang.Threadを利用し、その生成・実行開始及び停止を行う。

関連クラス：

- AmbientControl.class,
- AmbientControlImpl.class,
- AmbientDefaultDispatcher.class,
- AmbientThread.class,
- AmbientThreadGroup.class,
- AmbientStatus.class,
- AmbientTimeout.class

エージェント 間通信マネジャー コールバックメソッド マネジャー

親エージェントから子孫エージェントに対して呼び出すメソッドを実行する。

サービスメソッド マネジャー

子エージェントから親エージェントに対して呼び出すメソッドを実行する。

関連クラス：

- AmbientCallMessage.class,
- AmbientCallback.class,
- AmbientCallbackManager.class,
- AmbientMessageManager.class,
- AmbientMessageQueue.class,
- AmbientMethod.class

エージェント階層マネジャー

エージェントの能動実行を実現する。能動性は Java 言語の java.lang.Thread を利用し、その生成・実行開始及び停止を行う。

コールバックメソッドマネジャー

親エージェントから子孫エージェントに対して呼び出すメソッドを実行する。

サービスメソッドマネジャー

子エージェントから親エージェントに対して呼び出すメソッドを実行する。

関連クラス：

- AmbientContainer.class,
- AmbientIdentifier.class

2.3 協調スクリプティング言語処理系

モーバイルエージェントの階層構造は、構成要素のエージェントや集合全体の移動を通じてその構成や計算環境が変化する。このため、オブジェクト指向計算でみられるような静的な関係でモーバイルエージェントを結びつけることはできない。そこで、エージェントの合成・協調方法を規定するスクリプティング言語を導入する。既存のエージェント記述言語は、個々のエージェントの通信・動作をターゲットとしているのに対し、この研究では、個々のエージェントの動作記述はエージェントプログラムで与えることとし、このスクリプティング言語は、エージェント間の結合関係や通信、リソース情報などの各種サービスをメタ定義するものとなる。なお、このスクリプティング言語は、その意味論を並列計算理論であるプロセス代数、その中でも R.Milner の π 計算や L.Cardelli の Mobile Ambients を設計背景におき、実際の記述では通信環境との親和性から URL をベースとして拡張言語を元にする。

協調スクリプティング言語処理系は、Java 言語のプログラムであり、前述のエージェントランタイムシステムに組み込まれて実装されている。

スクリプティング言語ランタイム

スクリプティング言語の解釈実行をする部分

関連クラス：

- AmbientRuntime.class,
- AmbientRuntimeImpl.class,

エージェント階層制御

スクリプティング言語の記述に従ってエージェント階層を制御する。

関連クラス：

- AmbientIdentifier.class,
- AmbientInfo.class,
- AmbientURL.class,
- AmbientURLConnection.class

協調スクリプティング言語のユーザインターフェース

協調スクリプティング言語に記述の入力及び実行結果の表示部分

関連クラス：

- AmbientActionControl.class,
- AmbientButton.class,
- AmbientDestDialog.class,
- AmbientDialog.class,
- AmbientMenu.class,
- AmbientMonitor.class,
- AmbientTableControl.class,
- AmbientTableEvent.class,
- AmbientTableListener.class,
- AmbientTableMap.class,
- AmbientTableModel.class,

- AmbientTableSorter.class,
- AmbientTableView.class,
- AmbientTreeControl.class,
- AmbientTreeEvent.class,
- AmbientTreeListener.class,
- AmbientTreeNode.class,
- AmbientTreeView.class

2.4 高階モーバイルエージェントのフレームワークライブラリ

各エージェントの記述において利用するプログラム群であり、エージェント階層内の移動に関わる API 群、異なるコンピュータ間の移動に関わる API 群、エージェントの永続化や複製に関わる API 群、ランタイムシステムや他のエージェントに関するの情報を提供するための API 群から構成され、エージェントに対してはコンテキスト構造対して提供される。

エージェント階層内の移動に関わる API 群

エージェントがランタイムシステム内、つまり一つの階層内で移動する際に利用する API である。

関連クラス：

- Context.class,
- AmbientContextImpl.class,
- AmbientEvent.class,
- AmbientEventDispatcher.class,
- AmbientFuture.class,
- AmbientMessage.class,
- AmbientMethod.class

異なるコンピュータ間の移動に関わる API 群

エージェントが異なるランタイムシステムにわたる移動を行う際に利用する API である。

関連クラス：

- AmbientRequest.class,
- AmbientRequestFactory.class,
- AmbientRequestParser.class,
- AmbientResponse.class,
- AmbientResponseFactory.class,
- AmbientResponseParser.class,
- AmbientMATPFactory.class,
- AmbientMATPHandler.class,
- AmbientMATPRequest.class

エージェントの永続化や複製に関わる API 群

エージェントを永続化する際に利用する API である。

- Context.class
- AmbientContextImpl.class
- AmbientByteCode.class,

情報を提供するための API 群

エージェント及びランタイムシステムの情報、例えばエージェントの名前や存在位置を取得するための API である。

- Context.class
- AmbientContextImpl.class

第3章　プログラムの機能

ここでは MobileSpaces システムの機能を概説する。このシステムは次の 3 つの部分から構成される。

- 高階モーバイルエージェントのランタイムシステム (MobileSpaces ランタイムシステム)
- 協調スクリプティング言語処理系 (Mobile Agent URL インタプリター)
- モーバイルエージェントのフレームワークライブラリ (MobileSpaces エージェントコンテキスト)

3.1　高階モーバイルエージェントの実現機能

高階モーバイルエージェントは MobileSpaces ランタイムシステムにより実現される。高階モーバイルエージェント及びエージェントの動的合成・置換を実現するためのランタイムシステムである MobileSpaces ランタイムシステムが提供する機能及びその動作内容を概説する。

3.1.1　MobileSpaces のモーバイルエージェント モデル

MobileSpaces システムのエージェントモデルは以下の機能をモーバイルエージェントに拡張したものとなる。

- **エージェントの階層的構成:**　モーバイルエージェントはその内部に 0 個以上の別のモーバイルエージェントを入れ子状に内包できるとする。また、内包されたエージェントもそれぞれ能動的に動作する。
- **エージェント 間移動:**　モーバイルエージェントはコンピュータだけでなく、他のモーバイルエージェントに移動できるとし、そのとき、それに内包されたモーバイルエージェントもその階層構造を保ったまま移動するとする。

ここでエージェント階層間の関係を述べる。エージェント階層において、各エージェントは一つ外側に位置するエージェントが提供する各種リソース及びサービスを利用できる。一方、外側エージェントはそれが内包するエージェント階層上の任意のエージェント

に対して生成、停止、直列化などの実行制御を行うことができ、また、内包されたエージェントに用意されたコールバックメソッドを適宜呼び出すことができる。なお、内包エージェントが利用する各種サービスや呼び出されるコールバックメソッドは、外側エージェントにより規定される。このため、別のエージェントに移動することにより、利用可能なサービス内容や、コールバックメソッドの呼び出しタイミングや名前を動的に変更できるようになる。

3.1.2 ランタイムシステムの機能

ランタイムシステムが提供する機能について紹介する。

- エージェント階層の管理
- 高階エージェント実行の管理
- 高階エージェント実行の永続化
- 高階エージェントの移動
- 高階エージェントの直列化
- 高階エージェントの置換
- 高階エージェントの複製

コアシステムは計算環境への依存性を最小限にするため、計算環境に依存した機能は提供しない。このため、エージェントのコンピュータ間移動などの機能は、通信環境に依存することからシステム補助モーバイルエージェントにより提供される。

エージェント階層の管理

エージェント階層はエージェントの入れ子関係を基礎とすると木構造であり、各ノードが一つのモーバイルエージェントなる。コアシステムはそれぞれ一つのエージェント階層を維持管理し、そのコアシステム上のはエージェントの移動要求に対して、階層内の移動を実現する。なお、エージェント計算の整合性を保つため、エージェント階層は制限をもつ。

- コアシステムはそれが管理維持するエージェント階層の根となるステーショナリエージェントである。
- エージェント階層内において、各エージェントはそれの子孫エージェントの実行を制御することができる。つまり、子孫エージェントを他のエージェントに移動させたり、永続化や停止させることができる。

- また、エージェントはそれの先祖エージェントの実行を制御することはできない。ただし、エージェントはそれの親エージェントのメソッドを呼び出しを依頼することができる。ただし、親エージェント以外の祖先エージェントへのサービス要求はその祖先エージェントがステーショナリエージェントに限って可能である。

エージェント階層内のエージェント移動は、エージェント階層構造の変形として実現される。例えば、エージェントの移動は、そのエージェントの根とする部分木を、移動先のエージェントを根とする部分木の枝とすることである。なお、エージェント階層内の移動では、エージェントは能動的に動作したまま移動し、エージェントは明示的にイベントリスナーを登録することにより、移動発生時にはイベントとして通知される。

エージェント実行の管理

各エージェントは初期化、実行、永続化、停止の４つの実行状態をもち、その状態が実行中であるときは、複数の能動的スレッドをもつことができる。ただし、これらの実効状態や能動実行はコアシステムにより管理・制御される。例えば、エージェントが移動、永続化、停止される場合にはコアシステムにより能動実行が停止させられる。また、親エージェントのメソッドを呼び出しや、子孫エージェントへのコールバックメソッドでは、タイムアウトを課すことができるが、タイムアウトの管理はコアシステムにより実現する。

エージェント実行の永続化

エージェントはコンピュータ間移動する際には、ビットストリームに直列化される。この直列化とその逆変換はコアシステムにより提供される。コアシステムにより (非) 直列化を行うのは、不正なエージェントを検出を行うためである。

3.1.3　システム補助用モーバイルエージェント

エージェントに対する操作の一部は、各操作に対応したシステム補助用モーバイルエージェントを通じて実現される。これらはランタイムシステム上に用意されたエージェントであり、到着したエージェントに対して直ちに所定の操作を行う。つまり、操作の対象となるエージェントを、その操作を実現するシステム補助用エージェントに移動させると、その補助用エージェントが到着たエージェントに対してその操作を自動的に行うものである。

コンピュータ間移動

これはエージェントのコンピュータ間移送を実現するモーバイルエージェントであり、送信側 (移送元) と受信側 (移送先) の組として用意される。送信側エージェントは、それ

に到着したエージェントではエージェント A を直ちに直列化し、エージェントの実行状態とプログラムコードを受信側エージェントに送信する。受信側エージェントではその受信したデータを再びエージェントに戻す。なお、両エージェント間のエージェント移送において、TCP や UDP などのデータ転送プロトコルの選択や、移送における通信手順は両エージェントにより定義される。そして、別のコンピュータ間移送エージェントに移動することにより、移送方式を動的に変更できるようになる。なお、エージェント内部に子エージェントを内包する場合には、階層構造を保ったまま子エージェントも一緒に移動する。

永続化

これは内包したエージェントを 2 次記憶上に永続化するエージェントである。内包されるエージェントをこのエージェントに移動すると自動的に永続化される。なお、エージェントの永続化方法は、そのデータ形式などが多様化する可能性があり、永続化エージェントを適宜選択することにより永続化手法を選択する。なお、実行状態の直列化などはランタイムシステムのサービスにより実現する。なお、エージェント内部に子エージェントを内包する場合には、階層構造を保ったまま子エージェントも一緒に永続化される。

捕捉

一部の機能をモーバイルエージェントにより提供することにより次の利点がある。

- エージェントに対する各種操作を、移動というモーバイルエージェントの基本的な計算メカニズムとして統一的に取り扱うことができるため、エージェントプログラムの複雑性を軽減させることができる。
- 各種機能がモーバイルエージェントとして提供されるため、新たな機能を提供するモーバイルエージェントを移動させることにより、システムの動的拡張を行うことができる。
- モーバイルエージェントの利点の一つは、特定のコンピュータに束縛されずに実行できることにある。このため、エージェントに対するすべての操作をコンピュータ（ランタイムシステム）それぞれに用意する必要なく、特定のコンピュータにおいてのみ実現し、そこに移動して操作を行えばよい。
- コンピュータ（ランタイムシステム）もモーバイルエージェントとして扱えることから、携帯端末などの移動性をもつコンピュータについてもモーバイルエージェントとして統一的に扱える。

3.2 モーバイルエージェントの動的置換機構

MobileSpaces システムのランタイムシステム、さらに MobileSpaces システムのアプリケーションシステムはモーバイルエージェントの集合体として構成される。このため、新しいモーバイルエージェントをシステムに移動させることにより、そのエージェントによる機能が利用できるようになる。また、変更方法は後述のスクリプティング言語により記述され、計算環境やユーザ要求の変化に応じた変更が可能である。このほかに、エージェントの実況状態を保持した置換機構を提供する。

3.2.1 エージェントの置換判定・置換機構

MobileSpaces システムによるエージェントは、その実行中においてもその動作内容を動的に変更することができる。これはエージェントを実現する Java 言語オブジェクトの動的変更することにより実現される。ただし、置換対象となる二つのエージェントを構成する Java 言語オブジェクトが以下の 2 つの条件を満足する必要がある。

- それぞれがプライベート変数を含めて、同一の名かつ型のインスタンス変数を持つ。
- 置換後のエージェントは、置換前エージェントのエージェントプログラムが提供するコールバックメソッド及びエージェントコンテキストが提供するサービスメソッドのすべてを提供できる。

後者はエージェント間の協調関係を継承するためである。以下に置換までの処理を記述する。

> まず、置換する際には Java 言語のイントロスプクションにより、置換先と置換元の二つのエージェントがもつ Java 言語オブジェクトのメソッド名、データ型などをリスト化される。そして、置換後のエージェントが置換前エージェントと同じ名前のメソッドをもち、さらにそのデータ型が置換前エージェントのそれに対する同一またはサブクラスとなるときに置換可能と判定する。
>
> そして、置換可能なエージェントはまず直列化機能によりその状態がデータ化される。そして、置換後のエージェントのプログラムコードを通じて非直列化される。これにより、プログラムコードの置換を実現する。また、エージェントの名前や識別子は必要に応じて継承する。

置換可能性の判定は次の規則に基づく。置換元エージェントのエージェントプログラム及びエージェントのコンテキストの公開メソッドのメソッド m_i と、同じ名前のメソッドが置換先のエージェントに含まれるかを調べる。名前が一致する場合はさらにそのデータ型が一致性または代用可能性をサブタイプ関係を基準に判定する[1]。仕様側の m_i メソッ

[1] subclassing-is-subtyping に基づく。

ドのデータ型を $m_i : (A_i^1 \times \cdots A_i^n) \times \rightarrow A_i$ とし (A_i^j は j 番目引数の型、A_i は返値型)、到着エージェント側の m_i メソッドのデータ型を $m_i : (B_i^1 \times \cdots B_i^n) \times \rightarrow B_i$ とするとき、

$$A_i <: B_i \text{ かつ } B <: A$$

$C <: C'$ はデータ型 C は C' のサブタイプであることを表す。仕様側の m_i メソッドが上記の関係を満足するとき、置換可能エージェントとして判定する。

なお、置換可能性の判定では Java 言語の JDK に含まれる serialver プログラムによるクラス識別を利用し、高速化することができる。

3.3 高階エージェントのスクリプティング言語処理系

高階エージェントのスクリプティング言語は L.Cardelli と A.D.Gordon によって提案された Mobile Ambients をその意味論のベースとしている。

3.3.1 Mobile Ambients

Mobile Ambients は移動プログラムのための計算モデルであり、既存のプロセス代数／計算と同様な、操作意味論に基づく項書き換えシステムとして定式化される。ただし、既存のプロセス代数／計算がプロセス間通信を基本計算メカニズムにしているのに対し、Mobile Ambients は能動的プログラムの移動を基礎としている[2]。なお、Mobile Ambients は λ 計算のコーディング能力や各種の並列システムに対する優れた表現性がすでに明らかになっている。

Mobile Ambients の構文 (一部) は以下のようになる。

$$\begin{aligned} P \quad ::= \quad & n[P] \quad | \quad (\nu n)P \quad | \quad P_1 \,|\, P_2 \\ & | \quad \mathbf{in}\, n \,.\, P \quad | \quad \mathbf{out}\, n \,.\, P \end{aligned}$$

ここで P は Ambient と呼ばれる計算実体であり、プロセス代数／計算におけるプロセスに相当する。n は名前またはケーパビリティであり、$(\nu n)P$ は名前またはケーパビリティ n を P 内にカプセル化する。また、$P_1|P_2$ は P_1 と P_2 の並列に動作することを表す。ここで $P_1|P_2$ と $P_2|P_1$ の意味的な違いはない。

Ambient の移動（IN）： $\mathbf{in}\, n \,.\, P$ は名前 n をもつ Ambient に入ることを表し、例えば次のような遷移を行う。

$$n[\mathbf{in}\, m \,.\, P_1 \,|\, P_2] \,|\, m[P_3] \quad \rightarrow \quad m[n[P_1 \,|\, P_2] \,|\, P_3]$$

ここで、$n[\mathbf{in}\, m \,.\, P_1 \,|\, P_2] \,|\, m[P_3]$ は二つの Ambient の $n[\mathbf{in}\, m \,.\, P_1 \,|\, P_2]$ と $m[P_3]$ が並列動作することを表す。そして、前者は名前が n の Ambient の中で二つプログラム $\mathbf{in}\, m \,.\, P_1$

[2] π 計算 [?] における Mobility とは並行プロセス間の通信ポートの受け渡しを指し、プロセス自体の移動性は考慮されていないことに注意。

と P_2 が並列に動作することを表す。一方、後者は名前が m の Ambient の中でプログラム P_3 が動作していることを表す。そして、上記の遷移は、前者が $\mathbf{in}\, m\, .\, P_1$ により名前 n をもつ Ambient が、名前が m の Ambient の中に入ることを表す。

Ambient の移動 (OUT)： $\mathbf{out}\, m\, .\, P$ は 名前 m をもつ Ambient から出ることを表す。これの遷移例として $m[n[\mathbf{out}\, m\, .\, P_1 \mid P_2] \mid P_3]$ を考える。これは m という名前をもつ Ambient に二つの Ambient である $n[\mathbf{out}\, m\, .\, P_1 \mid P_2]$ と P_3 が入っている状態である。前者は名前 n をもつ Ambient を表し、その内部では二つのプログラム $\mathbf{out}\, m\, .\, P_1$ と P_2 が並列に動作することを示している。なお、$\mathbf{out}\, m\, .\, P_1$ は 名前 m をもつ Ambient から出ることを表す。

$$m[n[\mathbf{out}\, m\, .\, P_1 \mid P_2] \mid P_3] \quad \rightarrow \quad n[P_1 \mid P_2] \mid m[P_3]$$

上記の遷移では $\mathbf{out}\, m\, .\, P_1$ をもつ Ambient である $n[P_1 \mid P_2]$ が名前 m の Ambient から出ることになる。

既存のモーバイルエージェントの枠組みと比較したとき Mobile Ambient は次のような特徴をもつ。

エージェントの階層化：

Ambient は、エージェントと同様に能動的なプログラムであると同時に、Ambient の内部に一つ以上の Ambient を入れ子状に含むことができる。

エージェントの主体的移動：

Ambient は $\mathbf{in}\, n\, .\, P$ または $\mathbf{out}\, n\, .\, P$ を通じて主体的移動する。ただし、その Ambient 自体を含む外側の Ambient が移動するときは、階層構造を保持しながらそれとともに移動する。このため、主体的移動する場合を除いて、Ambient の計算環境、つまりそれの外側の Ambient が変化することはなく、計算の継続性が保証される。

3.3.2 方針

この高階エージェントのスクリプティング言語は Mobile Ambients の基本計算メカニズムを導入する。

高階モーバイルエージェント

モーバイルエージェントは能動的な計算実体であり、その内部に複数のモーバイルエージェントが入れ子状に内包できるとする。そして、モーバイルエージェントは自律的に他のモーバイルエージェントに移動することができる。またこのとき、その内部にエージェントを内包する場合はそれらのエージェントも同時に移動し、内包関係を保存する。

これにより、モーバイルエージェントは他のモーバイルエージェントに移動することを通じて、モーバイルエージェントのグループ化や、モーバイルエージェント合成体の

の構造変更を実現し、逆にエージェントがそれを内包するエージェントから出ることによりグループや合成体の分解を可能にする。

なお、ここで高階性と呼ぶのは、ここ提案するシステムはエージェントの単なる階層化でなく、エージェントの通信や制御方法、計算環境が、そのエージェントを内包する外側エージェントによりメタ定義が可能となるようにシステム構築されるからである。次にこの高階性にもとづく特徴について述べる。なお、以降ではモーバイルエージェントを単にエージェントと呼ぶことがある。

エージェントの制御・サービス

エージェントが受動的に移動・永続化されるときは、その前後にエージェントに用意された所定のコールバックメソッドが呼び出される。これにより、移動・永続化に際して処理を行うことができる。ただし、呼び出されるコールバックメソッドの名前、データ型、タイミングはそのエージェントを内包する外部エージェントによりメタ定義可能とし、システムよる固定化は行わない。

ケーパビリティ

エージェントの移動先としてエージェントを対象とすることができるが、任意のエージェントを移動可能にすることは、協調関係が維持できないエージェントも取り込む危険性がある。特に、内包されるエージェントの状態変更通知や制御に利用するコールバックメソッドを、到着したエージェントがもっていないと、内包する側である外側エージェントは、そのエージェントの制御ができなくなる。

各エージェントは、内包可能なモーバイルエージェントがもつべきメソッドの仕様（メソッド名、データ型など）を動的に登録できるようにする。そして、エージェントが到着したときは、この仕様を満たすモーバイルエージェントだけを内包し、それ以外のエージェントの移動要求は拒否する。

3.3.3 高階エージェントのスクリプティング言語

各エージェントは、グローバルな識別子と、階層構造に依存したローカルな識別子の二つをもち、後者は URL により記述される。例えば、some.where.com/agent1/agent2 では、some.where.com はエージェントの属するホストの名前であり、/agent1/agent2 とは親エージェント agent1 の子エージェントとなる agent2 を示している。なお、各エージェントはエージェント階層内の他のエージェントに、自分自身または子孫エージェントを移動させることができる。移動には migrate() API を利用する。

```
migrate(new AgnetURL("/agent1/agent2"));}
```

は/agent1/agent2"));上記の例は、これを実行したエージェントが自分自身をエージェント階層内の/agent1/agent2 となるエージェントに移動させることを意味する。ただ

し、他のコンピュータに移動する場合や、永続化する場合には、コンピュータ間移動や永続化機能を提供するエージェントによって実現する。下記の例では transmitter というエージェントに some.where.com/agent1/agent2 への移動を依頼する。

```
migrate(new AgentURL("transmitter:some.where.com/agent1/agent2"));
```

MobileSpaces では、コンピュータ間移動や永続化などの計算環境に依存したサービスは、各サービスごとにエージェントが用意されている。従って、これらのサービスを提供するエージェントを選択することより、計算環境の変化に柔軟に対応することができる。一方、計算環境の変化は、MobileSpaces のランタイムシステムではなく、オペレーティングシステムなどの下位レベルのソフトウェアにより監視・通知される。そこで、オペレーティングシステムと MobileSpaces のインターフェースを導入する。ただし、MobileSpaces は多様なオペレーティングシステムやハードウェアで稼働することを目的としているため、オペレーティングシステムへの変更は必要ない方法とする。

エージェントの指定する URL は形式$(variable) で書かれた変数を含めることができる。この変数はオペレーティングシステムやシェルによって提供される環境変数に対応づけられる。これは動的環境変数 [7] 及び動的 URLs [9] と同様なものとなる。

```
$(TRANSMITTER)://some.where.com/agent1/agent2
```

上記では TRANSMITTER は環境変数に対応づけられた変数であり、その中身はコンピュータ間移動サービスを提供するエージェントの名前となる。そして、計算環境の変化に応じて、コンピュータ間移動に最適なエージェントの名前がその内容となる。

3.4 モーバイルエージェントのフレームワークライブラリ

3.4.1 モーバイルエージェント

モーバイルエージェントは Java 言語のオブジェクトであり、主な要素は次の通りとなる。

- **エージェントプログラム**
  各エージェントの動作を記述する Java 言語のプログラムであり、外部エージェントからの呼び出されるコールバックメソッドと、内包エージェントが呼び出すサービス用のメソッドから構成される。各メソッドプログラムには、外部エージェントへのサービス要求や内包エージェントへのコールバックメソッド 呼び出しを含むことがある。
- **エージェント コンテキスト**
  内包されるエージェントに対して各種サービス、例えば、生成、停止、エージェント間移動などを提供するための API 群からなり、エージェント識別子や階層構造を含むリソース情報も提供する。なお、外側のエージェントが提供するサービス用メソッドも、エージェントコンテキストを介して呼び出す。

- **内包エージェント**
  あるエージェントに内包されている 0 個以上のエージェントであり、能動性をもち、さらにこれの内部に入れ子状にエージェントを含むことがある。

この他、エージェントは、それが属している階層構造に従って与えられる相対的な識別子と、実行に必要なプログラムコードを保持する。

3.4.2 エージェントプログラム

MobileSpaces では、モーバイルエージェントのプログラムは、一つまたは複数の Java のクラスから構成される。ただし、そのうち一つ以上のクラスはクラス Ambient を親クラスとして継承する必要がある。なお、以降ではエージェントを定義するためのクラスをエージェントクラス、そのインスタンスを単にエージェントと呼び。

```
package mobilespaces.system;
import mobilespaces.*;
import mobilespaces.system.*;
import mobilespaces.system.event.*;
public interface Ambient {
  public Context getContext();
  public Enumeration getListeners();
  public void addDefaultListener(AmbientDefaultListener listener);
  public void removeDefaultListener(AmbientDefaultListener listener);
  public void addMobilityListener(AmbientMobilityListener listener);
  public void removeMobilityListener(AmbientMobilityListener listener);
  public void addPersistencyListener(AmbientPesistencyListener listener);
  public void removePersistencyListener(AmbientPersistencyListener listener);
  public void addStructureListener(AmbientStructureListener listener);
  public void removeStructureListener(AmbientStructureListener listener);
}
```

以下はエージェントプログラムの例である。

```
public class Simple extends Ambient implements DefaultEventListener {
  public Simple() {
    addDefaultListener(this);
    addChildrenContext(new BaseContext);
  }
  void create(AgentURL url) {
    System.out.println("I am just born.");
  }
```

```
  void destroy(AgentURL url) {
    System.out.println("I am dying.");
  }
  void serialize() {
    System.out.println("I am serialized");
  }
  void unserialize() {
    System.out.println("I am unserialized");
  }
  void add(AgentURL child) {
    System.out.println("An agent "+child+" is coming into mine");
  }
  void remove(AgentURL child) {
    System.out.println("An agent "+child+" is leaving from mine");
  }
  void leave(AgentURL dst) {
    System.out.println("I am going to "+dst);
  }
  void arrive(AgentURL dst) {
    System.out.println("I arrive "+dst);
  }
}
```

3.4.3 コールバックメソッド

エージェントクラス（ブートクラス）に含まれる、変数、メソッド名や種類には特に決まりはないが、生成、消滅、移動、到着などの状態変化した際に呼び出されるコールバックメッソドはいくつかのリスナークラスに規定されている。エージェントはそれらのリスナークラス内のメソッドを実装し、そのクラスのインスタンスを`addxxxListener(AmbientxxxListener listenr)`により、リスナーオブジェクトとして登録する。例えば、上記ではエージェント自信が、生成直後、消滅直前、移動直前、移動直後、直列化前、直列化後する際には、`addxxxListener(AmbientxxxListener listenr)`にインターフェースが定義されたコールバックメソッド `create()`、`destroy()`、`leave()`、`arrive()`、`serialize()`、`unseralize()`が呼び出され、他のエージェントがそれに移動してきたときはその直後に`add()`が、そして、子エージェントが離れる直前に`remove()`が呼び出される。

なお、各リスナークラスにはインターフェース定義されたメソッドをヌルメソッドとして定義したアダプタークラスが用意されている。これらを利用することにより、エージェントプログラムがそのメソッドを定義する必要がなく、また、メソッド呼び出し時の処理を変更したいときは、そのメソッドをオーバーライトすればよい。

DefaultListener のコールバックメソッド

表: DefaultListener のコールバックメソッド

コールバックメッソド名	引数の数	引数の型	返値の型
create()	0	void	void
destroy()	0	void	void
leave()	0	void	void
arrive()	0	void	void
serialize()	0	void	void
unserialize()	0	void	void
add()	0	void	void
remove()	0	void	void
duplicate()	0	void	void
child()	1	AgentIdentifier	void
parent()	1	AgentIdentifier	void

エージェント生成直後

```
void create() {
/* do something */
}
```

エージェントを生成した直後に呼び出される。エージェントの初期設定などを行うことができる。エージェントのライフタイム内では一度だけ呼び出される。

エージェント終了直前

```
void destroy() {
/* do something */
}
```

エージェントが、消滅する直前に呼び出される。エージェントの終了時の後処理などをここに書く。エージェントのライフタイム内では一度だけ呼び出される。なお、子エージェントを含む場合には子エージェントに対しても呼び出される。

エージェント移動直前

```
void dispatch(URL url) {
/* do something */
}
```

エージェントが他のコンピュータまたは他のエージェントに移動するときに、現在のエージェントを離れる直前に呼び出される。なお、他のコンピュータに移動するときは、エージェントは直列化される必要があり、dispatch() の実行後にコールバックメソッド serialize() が呼び出される。なお、子エージェントを含む場合には子エージェントに対しても呼び出される。引数 url には移動先の URL アドレスが入る。

エージェント 移動直後

```
void arrive() {
/* do something */
}
```

エージェントが他のコンピュータまたは他のエージェントに移動するときに、移動先のエージェントに到着した直後に呼び出される。なお、他のコンピュータに移動するときは、エージェントは非直列化する必要があり、arrive() の実行前にコールバックメソッド unserialize() が呼び出される。なお、子エージェントを含む場合には子エージェントに対しても呼び出される。

エージェント 直列化直前

```
void serialize() {
/* do something */
}
```

エージェントのコンピュータ間移動や永続化に際しては、エージェントをその実効状態を含めてビット列に変換する、つまり直列化する必要があるが、直列化の直前には上記のメソッドが呼び出される。なお、子エージェントを含む場合には子エージェントに対しても呼び出される。

なお、Java 言語の Serialization 機能を使って直列化することから、エージェントが直列化不可能なオブジェクトが参照しているときは、移動する前にその参照を禁止する必要がある。また、スレッドに関する情報も直列化の対象にはならない。

エージェント 直列化直後

```
void unserialize() {
/* do something */
}
```

エージェントのコンピュータ間移動や永続化に際しては、エージェントは直列化され、ビット列に変換されるが、それを再びエージェントに戻す、つまり非直列化する必要があるが、その直後には上記のメソッドが呼び出される。なお、子エージェントを含む場合には子エージェントに対しても呼び出される。

エージェント 複製直前

```
void duplicate() {
/* do something */
}
```

これはエージェントの複製が生成される直前に呼び出される。エージェントの複製では移動や永続化と同様にエージェントの実行状態を直列化される。このため、duplicate()の実行後にコールバックメソッド serialize() が呼び出される。なお、子エージェントを含む場合には子エージェントに対しても呼び出される。

エージェント 複製直後１

```
void parent(AgentIdentifier aid) {
/* do something */
}
```

エージェントの複製が生成された際に呼び出されるメソッド。ただし、複製により、同じエージェントが二つ存在することになるが、このメソッドはその内の一方で呼び出される。そして、引数にはもう一方のエージェントの識別子が入いる。

エージェント 複製直後２

```
void child(AgentIdentifier aid) {
/* do something */
}
```

エージェントの複製が生成された際に呼び出されるメソッド。ただし、複製により、同じエージェントが二つ存在することになるが、このメソッドはその内の一方で呼び出される。そして、引数にはもう一方のエージェントの識別子が入いる。

3.4.4 エージェント コンテキスト

エージェントコンテキストは以下の Context インターフェースを実装した Java 言語オブジェクトであり、エージェントとランタイムシステムとのインターフェースを提供する。例えば、ランタイムシステムや他のエージェントに関するの情報、エージェントのコンピュータ間移動、永続化、複製要求の受け付けなどのサービスをランタイムシステムに代わって提供する。

```
public interface Context {
  public abstract void setContainer(AmbientContainer ac);
  public abstract URL getCurrentURL();
```

```
  public abstract URL getParent(URL url)
      throws NoSuchLocationException;
  public abstract Enumeration getChildren();
  public abstract Enumeration getChildren(URL url)
      throws NoSuchLocationException;

  public abstract URL create(String name);
  public abstract void destroy(URL obj);
  public abstract void destroy();

  public abstract void migrate(URL dst, Message msg)
      throws NoSuchLocationException, NoSuchMethodException;

  public abstract void dispatch(URL obj, URL dst)
      throws NoSuchLocationException;
  public abstract void dispatch(URL dst)
      throws NoSuchLocationException;

  public abstract Object getService(Message msg)
      throws NoSuchServiceException;
  public abstract boolean hasService(Message msg);

  public abstract void getCallback(Message msg)
      throws NoSuchCallbackException;
  public abstract boolean hasCallback(Message msg);
  public abstract boolean hasCallback(URL url, Message msg)
      throws NoSuchLocationException;
  public abstract void getCallback(URL url, Message msg)
    throws NoSuchCallbackException, NoSuchLocationException;
}
```

3.4.5 エージェントコンテキストの取得

エージェントコンテキストはエージェントのスーパークラスである Ambient により提供されるメソッド getAgentContext() を通じて取得される。

```
Context getContext();
```

ただし、エージェントコンテキストはエージェントが移動または保存した際には有効ではなくなり、到着または復活した際に再び作られる。このため、到着または復活したときは、getAgentContext() を通じて再び取得する必要がある。

例：エージェント コンテキスト の取得

```
public class Hello extends Ambient {
  ....
  public void create() {
    AgentContext ac = getAgentContext();
    ....
  }
  ....
}
```

エージェントコンテキストを取得するときは、エージェントコンテキストが必要となる直前に上記のＡＰＩを実行し、また、エージェントコンテキストを保持するときは、メソッド変数に保持し、インスタンス変数やクラス変数には保持しないようにする。

3.4.6 エージェント コンテキスト の API

エージェントコンテキストの API は「各種情報の取得」、「エージェント間通信」、「状態変更」の３種類に大別される。代表的な API を示す。

- `public URL getCurrentURL()`
  エージェントが属しているコンピュータ名とエージェント階層中の位置を URL 形式で返す。

- `public URL getParentURL()`
  それの親エージェントが属しているコンピュータ名とエージェント階層中の位置を URL 形式で返す。

- `public Enumeration getChildren()`
  そのエージェントに属している子エージェントを列挙する。

- `public URL create(String name) throws NoSuchFileException`
  初期化前エージェントを読み込み、エージェントとして生成する。そしてその生成エージェントの URL を返す。

- `public void destroy(URL url) throws NoSuchAgentException`
  `url` により指定されたエージェントを終了させる。終了対象となるエージェント `url` が存在しないときは例外 `NoSuchAgentException` が発生する。

- `public void migrate(URL dst) throws NoSuchLocationException`

エージェントを dst に示されたエージェントに移動させる。移動先のエージェントが存在しないときは例外 NoSuchLocationException が発生する。

- public void migrate(URL url, URL dst) throws
  NoSuchLocationException,
  NoSuchAgentException

  url により指定されたエージェントを dst に示されたエージェントに移動させる。移動対象となるエージェント url が存在しないときは例外 NoSuchAgentException が発生し、移動先のエージェントが存在しないときは例外 NoSuchLocationException が発生する。

- public byte[] serialize(URL url) throws
  NoSuchAgentException

  url により指定されたエージェントを直列化し、ビット列に変換する。直列化対象となるエージェント url が存在しないときは例外 NoSuchAgentException が発生する。

- public AgentIdentifier unserialize(byte[] data) throws,
  NoSuchAgentException,
  IlleagaDataException

  data に直列化されたエージェントをその内部エージェントを含めて再びエージェントに戻し、これを実行したエージェントの内部に配置する。エージェント正しい直列化ビット列でないときは例外 IlleagaDataException が発生する。

- public AgentIdentifier unserialize(byte[] data, URL url) throws
  NoSuchAgentException,
  IlleagaDataException

  data に直列化されたエージェントをその内部エージェントを含めて再びエージェントに戻し、url により指定されたエージェント内部に配置する。エージェント url が存在しないときは例外 NoSuchAgentException が発生する。また、正しい直列化ビット列でないときは例外 IlleagaDataException が発生する。

- public void getService(Message msg) throws
  NoSuchAgentException,
  NoSuchMethodException

  クラス Message を通じてメソッド名、引数により指定した親エージェントのコンテキスト内のメソッドを呼び出す。メソッドがないときは、例外 NoSuchMethodException を発生し、親エージェントがないときは例外 NoSuchAgentException が発生する。

- public void getCallback(Message msg) throws
  NoSuchAgentException,
  NoSuchMethodException

クラス Message を通じてメソッド名、引数により指定した親エージェントのコールバックメソッドを呼び出す。メソッドがないときは、例外 NoSuchMethodException を発生し、親エージェントがないときは例外 NoSuchAgentException が発生する。

エージェントコンテキストの例を示す。

```
public class BaseContext extends Context {
  public void create(AgnetURL obj, AgentURL dst)
      throws ... { ... }
  public void destroy(AgentURL obj)
      throws NoSuchAgentException ... { ... }
  public byte[] serialize(AgentURL obj)
      throws NoSuchAgentException ... { ... }
  public URL unserialize(byte[] data, AgentURL dst)
      throws ... { ... }
  ....
}
```

メッセージクラス

ここでメソッド呼び出しに利用するクラス Message について説明する。MobileSpaces では、エージェントが別のエージェントの変数やメソッドを直接呼び出すことはできない。これは他のエージェントによる（不正な）呼び出しから守るためであるエージェント間で情報を交換する必要があるときには次の通信用のプリミティブを利用して通信をする。また、これらのプリミティブを通じて通常のメソッド呼び出しと比較して、柔軟で並列性を活かした通信を実現することができる。通信の方式には、同期メソッド呼び出し、非同期メソッド呼び出し、一方向の非同期メッセージ送信の 3 種類がある。

クラス Message の API は以下の通り

- コンストラク

  ```
  Message()
  ```

- コンストラク

  ```
  Message(String name)
  ```

- メッセージの引数の設定

  ```
  setArg(Object obj)
  ```

ここで setArg() の引数が通信メッセージの引数となる。引数は 0 個以上を設定することができ、さらに型に制限はない。ただし、引数にはいるオブジェクトは Java の Core API に含まれるクラスからのインスタンスでなければいけない。

呼び出される側では、メッセージ名と一致するメソッドに対して、setArg() で設定した順番の引数列として渡される。このため、受信側のエージェントに送信メッセージと同じ名前のメソッドがあり、さらにそれの引数の数とそれらの型がその順番を含めて一致する必要がある。例を示す。

```
Message msg = new Message(greeting);
msg.setArg(Hello);
msg.setArg(World);
```

この例は、greeting という名前のメッセージを生成するもので、その引数は、文字列型定数の Hello と World となる。なお、メソッド名がない場合や引数が一致しない場合は、ランタイムエラーとなる。

3.4.7 エージェント記述例

```
 1: public class Sample extends Ambient implements DefaultEventListener {
 2:   public Sample() {
 3:     addChildrenContext(new BaseContext);  // offering a context
 4:     addDefaultListener(this);             // registering itself as a listener
 5:   }
 6:   public void create(AgentURL url) {
 7:     try {
 8:       go(new AgentURL("transfer://some.where.com/agent1/agent2"));
 9:     } catch (MalformedURLException e) { ... }
10:   }
11:   public void arrive(AgentURL url) {
12:     System.out.println("arrived at "+url);
13:   }
14:   public void leave(AgentURL url) {
15:     System.out.println("leaving at "+url);
16:   }
17:   public void suspend() {}
18:   public void resume() {}
19:   ...
20: }
```

```
 1: public class Transfer extends Ambient implements DefaultEventListener {
 2:   public Transfer() {
```

```
 3:     addDefaultListener(this);              // registering itself as a listener
 4:     addChildrenContext(new BaseContext);   // offering a context
 5:     registry("transfer");                  // naming itself as transfer
 6:   }
 7:   public void add(AgentURL url) {    // invoked at having a new child agent
 8:     Message msg = new Message("serialize");
 9:     msg.setArg(url);                 // url specifies a new child agent
10:     byte[] data = (byte[])getService(msg); // serializing an agent specifed by url
11:     AgentURL dst = url.getTarget();  // dst specifies the original destination
12:     send_agent(data, dst);           // transmitting the serialized agent to dst
13:   }
14:   private send_agent(byte[] data, AgentURL dst) {
15:     // sending the serialized agent (data) to the destination (dst)
16:     ...
17:   }
18:   private receive_agent(byte[] data, AgentURL dst) {
19:     // invoked at receiving data for a remote Transfer
20:     Message msg = new Message("deserialize");
21:     msg.setArg(data);  // data is a serialized agent
22:     msg.setArg(dst);   // dst specifies the destination agent
23:     AgentURL url = (byte[])getService(msg); // deserializing data at dst
24:     ...
25:   }
26:   ...
27: }
```

3.5 MobileSpaces の実装

ここでは MobileSpaces システムの実装概要を述べる。MobileSpaces は Java 言語 [?] の仮想機械上で動作し、それのモーバイルエージェントは、コンピュータ間の移動性をもつ自己完備かつ能動的なプログラムであり、Java 言語のオブジェクトとして実現される。エージェント移動では、プログラムコード 及び実行状態とともに移動し、移動先コンピュータでは移動直前の状態から処理を継続することができる。本節では、モーバイルエージェント計算の最も基礎となるエージェント移動を中心に MobileSpaces システムの特徴を述べる。

3.5.1 エージェント移動の高速化

エージェント移動では、エージェントの実行状態とプログラムコードが移動するが、既存モーバイルエージェントシステムの多くは、状態転送後にコード (クラスプログラム) をオンデマンドに転送する。つまり、移動先でバインド要求が発生する毎にクラスプログラムの転送要求を送信し、そのクラスプログラムを移動先に転送させる。このため、クラスプログラム数に従って通信回数が増加し、エージェント移動コストが増大する。さらに、Java 言語はクラスプログラムを動的にバインドするため、クラスプログラムの移動タイミングは不確定となり、エージェントの状態移動後であっても、通信接続が断たれると、必要なクラスプログラムの転送が不可能となり、計算処理が継続不能となることがある。

そこで、MobileSpaces のエージェント移動では、そのエージェントの実行に必要なクラスプログラムのうち、移動先に存在しないクラスプログラムを、エージェントの状態とともに一括転送させる。このため、移動端末や無線通信などの常時接続が仮定できない通信ネットワークにおいても、エージェントの非同期実行が可能になる。

3.5.2 データ通信プロトコルへの非依存性

既存のモーバイルエージェントシステムの多くは、TCP や RMI などの特定のデータ通信プロトコルを利用して、エージェント移動させる。このため、これらの通信プロトコルが利用できない環境ではエージェント移動が実現できないという問題がある。そこで、MobileSpaces では特定のデータ通信プロトコルを仮定せず、信頼性のあるデータ通信プロトコルであれば、エージェント移動が可能であるように汎用的な構成とする。また、上述のように一括転送とすることにより、一方向一回のデータ転送でもエージェント移動が実現できるようになり、双方向通信ネットワークだけでなく、単方向通信ネットワークにおいても運用可能となる。これはファイル共有を利用したエージェント移動を可能にすることから有用となる。

3.5.3 通信切断を考慮したエージェント移動

既存のモーバイルエージェントシステムが提供するエージェント移動は、移動元コンピュータと移動先コンピュータが通信接続されていることが前提となっている。しかし、移動端末からネットワークを介して別の移動端末にエージェントが転送する場合、移動端末の非接続性により、両移動端末間の通信経路全体が同時に接続されているとは限らない。

そこで、部分的通信切断を考慮したエージェント移動機構を導入する。これは、ネットワーク内の移動経路上に中継ホストを用意し、中継ホスト間の通信接続が断たれている場合は、到達可能な中継ホストにエージェントを移動させ、そこに待機させる。そして、通信接続が回復した後に移動を再開させることを繰り返して、最終的に移動先に到

達させるものである。

3.5.4 エージェントの実行管理

各モーバイルエージェントはランタイムシステムの制御のもとで実行される。ランタイムシステムは、エージェントが生成、永続化、活性化、停止などの状態変化する前後に、エージェント側に用意された所定のコールバックメソッドを呼び出す。また、ランタイムシステムは、各エージェントに一つ以上のスレッドが割り当てる。ランタイムシステムは必要に応じてエージェントの強制終了や例外処理を行うことができる。

3.5.5 エージェントの移動

各エージェントのコンピュータ間移動はランタイムシステムにより実現される。

- エージェントの状態（エージェントプログラムのインスタンス変数）の直列化及びその逆変換
- エージェントのプログラムコードと直列化された状態の転送

エージェントの実行状態の直列化には Java 言語の ObjectSerialization 機構を利用する。このため、インスタンス変数は直列化対象となるが、メソッド変数やプログラムカウンタは除外される[3]。しかし、エージェントでは移動や永続化の前後に所定のコールバックメソッドが呼び出されるため、このコールバックメソッド中にインスタンス変数以外の状態保持と、ファイルなどの計算リソースの解放・確保が可能となる。

エージェントの直列化形式

エージェント転送に用いるエージェントのデータ形式について概説する。その形式は、ヘッダフィールド、コードフィールド、コンテキストフィールドの３つ部分からなる。

ヘッダーフィールド　MobileSpaces システムによるエージェントであることを示す識別子と、移動元の MobileSpaces システムのバージョン番号が示される。このほか、エージェントが暗号化されている場合は、その暗号化手法の名称を伴うことがある。これらはエージェント到着時にランタイムシステムにより最初に解釈される。

コンテキストフィールド　これは、エージェントの実行状態や個々のエージェントに関する情報を格納するフィールドであり、例えば表 1 のようなエントリをもつ。格納する際には各エントリ毎にタグを定めている。

[3] メソッド変数やプログラムカウンタはエージェント移動の対象とはならないが、[8] などで議論されているように、実際的なモーバイルエージェント応用ではインスタンス変数だけを直列化することで十分である。

表 1:コンテキストフィールドのエントリ (一部)

エントリ名	タグ	補足
識別子	AGID	64bit の全域的識別子
名前	NAME	エージェントの論理名
実行状態名	STAT	エージェントの実行状態名
実行状態	CNTX	エージェントプログラムの状態
子エージェント	CHLD	子エージェント
親エージェント	PRNT	親エージェント
生成ホスト	OGHT	生成されたホスト
移動元ホスト	PVHT	移動元ホスト
移動先ホスト	TGHT	移動先ホスト

コードフィールド　0 個以上の Java クラスファイルが格納される。また、実行に必要なファイル等が含まれることがある。

留意点

- エージェント移動における通信時間を短くするため、上述のコードフィールドとコンテキストフィールドは zip 形式で圧縮される[4]。ただし、コードフィールドは実行・移動過程で変更されないのに対し、コンテキストフィールドは変更可能性をもっている。そこで、コンテキストフィールドとコードフィールドは別々に圧縮し、一方、前者は移動するごとに圧縮されるのに対し、後者はエージェント生成時に一度だけ圧縮され、移動時には圧縮対象とはしない。これにより、圧縮範囲が最小化され、圧縮に要する計算時間を短くする。
- 一般にオブジェクト／プログラムの永続化などでは、それの永続化時と活性化時では、永続化を行うシステムと活性化するシステムが相違することがある。MobileSpaces のランタイムシステムは未定義のタグは無視する。このため、ランタイムシステムは必要に応じてエントリの拡張をしても互換性を維持される。

エージェント送受信

エージェントのコンピュータ間移動における手順を示す。

エージェント送信　エージェントはランタイムシステムを通じて、それ自身または子エージェントを他のコンピュータに移動させることができる。エージェントの移動要求があったときの手順を示す。

1. 移動対象となるエージェントがもつ所定のコールバックメソッドを呼び出す。
2. エージェントに割り当てられたすべてのスレッドの実行を停止するとともに、エージェント管理テーブルから削除する。

[4]圧縮後のサイズは 60 パーセント程度に減少される。

3. エージェントのインスタンス変数をネットワークに移動可能なデータ形式に変換（直列化）し、コンテキストフィールドに格納する。
4. エージェント識別子、名前、所有者、認証文字列などの管理情報を直列化し、コンテキストフィールド上の対応するエントリに登録する。
5. コンテキストフィールドを圧縮するとともに、ヘッダフィールドと圧縮済みのコードフィールドを合成する。
6. 移動先コンピュータに転送する。

ただし、直列化形式に直されたエージェントを移動先コンピュータに転送する方法は、エージェント移動に利用するデータ通信方法に依存する。例えば、TCP/IP を利用する場合には、移動先と TCP ソケットを接続し、それを通じて直列化形式に直されたエージェントを送信する。転送終了後はソケットを閉じる。また、ファイル共有を利用して転送する場合は、所定のディレクトリなどに直列化形式に直されたエージェント書き出し、それを移動元が読み取ることにより実現される。

エージェント受信　ランタイムシステムはエージェント移動に利用する通信チャネルを監視しており、データ到着後は以下の処理を行う。

1. エージェントを直列化したビット列を受け取る。
2. ヘッダファイルを判別後、コードフィールドとコンテキストフィールドを非圧縮状態に戻す。ただし、圧縮されたコードフィールドはそのまま保持する。
3. コンテキストフィールド内の各エントリを取り出し、エージェントの識別子や名前、所有者をエージェント管理テーブルに登録する。
4. 到着したエージェントのために名前空間を生成し、コードフィールド内の各クラスファイルを動的ロード可能な状態で保持する。
5. 直列化された状態をもとにエージェントのロードする。
6. エージェント到着後処理に対応するコールバックメソッドを呼び出す。

MobileSpaces はエージェント移動を完了を判定する方法を定めていない。これはエージェント移動に TCP などの信頼性のあるデータ通信プロトコルを利用することが前提となっており、これらのデータ通信プロトコルを介して、移動失敗が判別できるためである。ただし、移動先ホストは明示的に移動元に受信確認メッセージを送ることができる。

なお、データ通信プロトコルなどにより移動失敗が判定された場合は、所定時間後に複数回の再移動を試みる。ただし、再移動も失敗となる場合は、エージェントの所定コールバックメソッドを通じて、移動失敗をエージェントに通知し、移動元で実行が再開される。

3.5.6 エージェント間通信

MobileSpaces は同一ランタイムで実行されるエージェントは互いに通信を行うことができる[5]。これは、他のエージェントのメソッドを呼び出すことにより実現されるが、その返値の返し方により、以下の 3 つの形態がある [?]。

- 非同期メッセージ送信
- 同期メソッド呼び出し
- フュチャー通信

呼び出されるメソッドは、メッセージの名前を同じ名前をもつプブリックメソッドであり、さらにメッセージの引数の型はメソッドの仮引数は一致またはサブクラスとなるものである。対応するメソッドがない場合は例外が発生する。また、タイムアウト機構により、送信後所定時間が経過しても対応したメソッド実行が完了しない場合は、タイムアウト例外が送信側に通知され、そのメソッドの実行が開始されているときはその実行は中断される。

なお、MobileSpaces は明示的に複数スレッドを実行しない限り、各エージェントがもつ実行スレッド数は高々一つであり、一度に一つのメッセージしか処理されない。このため、各エージェントはメッセージキューをもっており、他のメッセージまたはコールバックメソッドを実行中に到着したメッセージは、メッセージキューに格納され、その到着順に処理される[6]。

非同期メッセージ送信　送信側エージェントはメッセージ送信後は、ブロックされることなく実行を継続することができる。一方、受信側エージェントでは対応するメソッドを呼び出されるが、その返値は無視される。

同期メソッド呼び出し　送信側エージェントはメッセージを送信し、そのメソッド呼び出しが完了するまでブロックされる。一方、受信側エージェントでは対応するメソッドを呼び出される。なお、必要に応じてメソッド呼び出し終了時間に時間制限を課すことができる。

フュチャー通信 (非同期メソッド呼び出し)　送信側エージェントはメッセージを送信すると同時に、値を受け取るためのフュチャーオブジェクトを生成し、返答を待たずに処理を継続する。受信側エージェントでは対応するメソッドを呼び出され、それの返値をフュチャーオブジェクトに返す。送信側はフュチャーオブジェクトを読み出すことにより、メッソドからの返値を得る。ただし、読み出した時点でメソッド呼び出しが完了していない場合は、フュチャーオブジェクトが返値を受け取るまでブロックされる。同期メソッド呼び出しと同様にタイムアウトにより時間制限を設けることができる。

[5]相違なランタイム間の通信は提供しない。これはエージェント移動により実現できるためである。
[6]デッドロックを回避するため、エージェントそれ自身へのメッセージは優先的に実行される。

3.5.7　エージェントの永続化

ランタイムシステムはエージェントを2次記憶などに保存することができる。なお、オブジェクト永続化では、実行状態とコードを2次記憶などに保存することをさすが、モーバイルエージェントではそれ自身を移動できるため、それ自身の保存において、2次記憶だけでなく、常時稼働のサーバに移動することも永続化と等価に扱えることになる。そこで、エージェントの永続化は2次記憶を含めた場所へのエージェント移動としてとらえる。

この他、その永続化形式と移動中エージェントの直列化形式を同一なものにする。これにより、ネットワークを介した移動だけでなく、フラッシュカードやフロッピーディスクなどの記憶メディアの再マウントをエージェント移動として統一的に扱えるようになる。

3.5.8　エージェントの生成

ランタイムシステムは2次記憶に保存されるエージェントファイルから、エージェントを生成することができる。また、実行状態にあるエージェントと、同一の実行状態とコードを保持した別のエージェントを動的に生成することもできる。後者は付加分散などに有用となる。

第4章　プログラムの稼働環境

MobileSpaces システムは計算環境依存性を最小限にシステムが構成されている。このため、多様な計算環境で動作することが確かめられている。また、異なる稼働環境でもMobileSpaces システムが動作すればエージェントの移動が実現でき、高い相互運用性を持っている。

4.1　稼働環境：ハードウェア

Sun 社 JDK 1.1 相当以上 (JDK1.0.2 では動作しません) が動作するハードウェア上で動作する。現在、動作確認がなされているハードウェアは以下の通り

- IBM PC-AT 互換機
- Sun 社 Ultra Space Station
- Apple 社 PowerMac

4.2　稼働環境：オペレーティングシステム

Sun 社 JDK 1.1 相当以上 (JDK1.0.2 では動作しません) が動作するオペレーティングシステム上で動作する。現在、動作確認がなされているオペレーティングシステムは

- Microsoft 社 Windows95
- Microsoft 社 Windows98
- Microsoft 社 WindowsNT Workstation/Server version 4.0
- Microsoft 社 WindowsNT Workstation/Server version 5.0 β 2
- Sun 社 Solaris 2.5, 2.6
- Apple 社 MacOS 8.0, 8.5
- Linux
- FreeBSD

ただし、Linux と FreeBSD はスクリプティング言語のユーザインターフェースを省く。

4.3 稼働環境：支援ソフトウェア

- Sun 社 Java Development Kit (JDK) version 1.1.2, 1.1.3, 1.1.4, 1.1.5, 1.1.5, 1.1.7A, 1.1.7B
- Symantec 社 Visual Cafe version 2.0, 2.5, 3.0
- Microsoft 社 Visual J++ version 6.0
- Borland 社 jBuilder version 2
- Metrowerks CodeWarrior

ただし、Visual J++はスクリプティング言語のユーザインターフェースを省く。また、いずれも Sun 社 Java Foundation Class (Swing) ライブラリが必要である。

4.4 稼働環境：コンピュータ付属機器

TCP/IP に対応したネットワーク機器。

第5章　プログラムの操作

5.1　MobileSpacesシステムのインストール方法

インストールは次の３つの手順からなる。

(1) MobileSpacesアーカイブの解凍

(2) ディレクトリの展開

(3) 環境変数の設定

5.1.1　MobileSpacesアーカイブの解凍とディレクトリの展開

納品プログラム (FD) 中のディレクトリ構造を示す。

```
mobilespaces -|
              |- event -|
              |         |- *.class
              |
              |- monitor -|
              |           |- *.class
              |
              |- protocol -|
              |            |- http -|
              |            |        |- *.class
              |            |
              |            |- matp -|
              |            |        |- *.class
              |            |
              |            |- *.class
              |
              |- system -|
              |          |- *.class
              |
              |- zip -|
```

```
|       |- *.class
|
|- Context.class
|- Future.class
|- Message.class
|- NoSuchAmbientException.class
|- NoSuchCallbackException.class
|- NoSuchLocationException.class
|- NoSuchServiceException.class
|- Station.class
|- TimeoutException.class
```

5.1.2 環境変数の設定

MobileSpacesはJava言語の仮想機械上で動作する。このため、MobileSpaces関連のクラスを利用できるように環境変数CLASSPATHを設定する必要がある。これには上記ディレクトリ構成図におけるディレクトリmobilespacesをCLASSPATHのディレクトリパスに加える。また、もし、環境変数CLASSPATHに「.」(カレントディレクトリ)が含まれいるときは、ディレクトリmobilespacesをカレントディレクトリにすればよい。

5.2 MobileSpacesシステムの起動方法

２台以上のコンピュータで利用されることが推奨されるが、１台だけでも稼働は可能である。

5.2.1 起動方法(Windows95/98/NTの場合)

２台以上のコンピュータを利用する場合

各コンピュータで以下のコマンドを実行する。このときそれぞれのコンピュータの環境変数CLASSPATHに「．」があり、カレントディレクトリが上記ディレクトリ構成図におけるmobilespacesであるとする。

```
java  MobileSpaces.Station
```

それぞれのコンピュータで MobileSpaces システムが起動し、システム間でプログ ラムの移動が可能になる。なお、 MobileSpaces システムはTCP/IP通信ポートの 5000 番を使用する。もし、この 5000 番が他の通信サービスと衝突する場合はbindエラーとなる。この場合はポート番号を変更する必要がある。

```
java  MobileSpaces.Station 5001 5001
```

上記では TCP/IP ポート番号を 5001 とする。

5.2.2 一台のコンピュータだけで利用する場合

以下のコマンドを実行する。このときそれぞれのコンピュータの環境変数 CLASSPATH に「．」があり、カレントディレクトリが上記ディレクトリ構成図における mobilespaces であるとする。

```
start java  MobileSpaces.Station 5000 5001
start java  MobileSpaces.Station 5001 5000
```

第一引数はエージェントの受信用の TCP/IP 通信ポート番号、第二引数はエージェント送信用の TCP/IP ポート番号となる。上記の例では、二つの MobileSpaces システムが起動し、前者の 5000 番で受信し、5001 番で送信する。逆に後者は前者の 5001 番で受信し、5000 番で送信することになり、同じコンピュータ上でありながら二つの MobileSpaces システム間でエージェントの移動が可能になる。なお、この例では通信ポートの 5000 番と 5001 番を使用する。ただし、この 5000 番または 5001 番が他の通信サービスと衝突する場合は bind エラーとなる。

5.2.3 起動方法 (UNIX の場合)

２台以上のコンピュータを利用する場合

各コンピュータで以下のコマンドを実行する。このときそれぞれのコンピュータの環境変数 CLASSPATH に「．」があり、カレントディレクトリが上記ディレクトリ構成図における mobilespaces であるとする。

```
java  MobileSpaces.Station
```

それぞれのコンピュータで MobileSpaces システムが起動し、システム間でプログ ラムの移動が可能になる。なお、 MobileSpaces システムは TCP/IP 通信ポートの 5000 番を使用する。もし、この 5000 番が他の通信サービスと衝突する場合は bind エラーとなる。この場合はポート番号を変更する必要がある。

```
java  MobileSpaces.Station 5001 5001
```

上記では TCP/IP ポート番号を 5001 とする。

5.2.4 一台のコンピュータだけで利用する場合

以下のコマンドを実行する。このときそれぞれのコンピュータの環境変数 CLASSPATH に「．」があり、カレントディレクトリが上記ディレクトリ構成図における mobilespaces であるとする。

```
java  MobileSpaces.Station 5000 5001 &
java  MobileSpaces.Station 5001 5000 &
```

第一引数はエージェントの受信用の TCP/IP 通信ポート番号、第二引数はエージェント送信用の TCP/IP ポート番号となる。上記の例では、二つの MobileSpaces システムが起動し、前者の 5000 番で受信し、5001 番で送信する。逆に後者は前者の 5001 番で受信し、5000 番で送信することになり、同じコンピュータ上でありながら二つの MobileSpaces システム間でエージェントの移動が可能になる。なお、この例では通信ポートの 5000 番と 5001 番を使用する。ただし、この 5000 番または 5001 番が他の通信サービスと衝突する場合は bind エラーとなる。

5.3 MobileSpaces システムの操作方法

MobileSpaces システムが起動するとウィンドウが現れる。このウィンドウはエージェントの起動、移動、停止、永続化などを行うモニターである。ウィンドウ下部にあるボタン「Load」、「Destroy」、「Move」を押すことにより、エージェントの起動、停止、移動を実現する。

5.3.1 エージェントのロード

このウィンドウから「Load」ボタンを押し、ファイルリストから拡張子が「Editor.class」となるファイルを選択する。ここで現れるウィンドウはモーバイルエージェントのエディタープログラムであり、モーバイルエージェントフレームワークを利用して作成されているため高階モーバイルエージェントとして機能し、その内部に子エージェントを内包することができる。

5.3.2 エージェントの移動

MobileSpaces のウィンドウの中央にあるリスト中のエージェントを選択し、「Move」ボタンにより移動させることができる。

Name	Status	URL
Transfer	Normal	/
Editor	Normal	/
Printer	Normal	/

上記ようなリストにおいて、Editor エージェントを選択して、「Move」ボタンにより移動させる。このときダイアログにより移動先を入力することができる。また、URL ベースのスクリプティング言語プログラムも入力することができる。

Move To: /Printer

例えば、上記のように入力すると Editor エージェントは Printer エージェントに移動する。その結果、リストは以下のように変化する。

Name	Status	URL
Transfer	Normal	/
Editor	Normal	/Printer
Printer	Normal	/

また、コンピュータ間移動をさせるときは、エージェントのコンピュータ間移動を提供するモーバイルエージェントを利用する。ここで Transfer はプログラムの機能に関する説明で取り上げた Transfer クラスから生成されたエージェントであり、子エージェントに対してコンピュータ間移動機能を提供するものとなる。そこで、Transfer エージェントによりコンピュータ間移動を実現する。これには例えば次のように移動対象のエージェントを移動させる。

Move To: Transfer://some.where.com/agent1/agent2

5.3.3 エージェントの終了

MobileSpaces のウィンドウの中央にあるリストから終了させたいエージェントの項目をクリックして選択し、MobileSpaces のウィンドウの下段にある「Close」ボタンをクリックすると、その選択されたエージェントは終了する。

5.3.4 エージェントの永続化

MobileSpaces では、エージェントをその実行状態ごとファイルにとして保存することができます。MobileSpaces のウィンドウの中央にあるリストから保存させたいエージェントの項目をクリックして選択する。そして、MobileSpaces のウィンドウの下段にある「Save」ボタンをクリックする。保存するファイル名を聞いてきますのでファイル名を付けて下さい。ただし、ファイル名は「.class」で終わる。先のエージェントの起動の手順に従って、この保存ファイルを起動すればセーブした時点から再開することができる。

関連図書

[1] J.Baumann and N.Radounklis, *Agent Groups in Mobile Agent Systems*, Conference on Distributed Applications and Interoperable Systems, 1997.

[2] D. B. Lange, and M. Oshima, Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley, 1998.

[3] ObjectSpace Inc, ObjectSpace Voyager Technical Overview, ObjectSpace, Inc. 1997.

[4] I. Satoh, AgentSpace: A Mobile Agent System, http://islab.is.ocaha.ac.jp/agent/index.html, 1997.

[5] 佐藤一郎, 高階モーバイルエージェントシステム, 情報処理学会プログラミング研究会報告, 3 月, 1998.

[6] 佐藤一郎, et al, "モーバイルエージェントの階層的な構成と移動" 日本ソフトウェア科学会全国大会, September, 1998.

[7] B. N. Schilit, and N. Adams, and R. Want, Customizing Mobile Application, Proceedings of Workshop on Mobile Computing Systems and Applications, IEEE Press, 1993.

[8] M. Strasser and J. Baumann, and F. Hole, Mole: A Java Based Mobile Agent System, Proceedings of ECOOP Workshop on Mobile Objects, 1996.

[9] G. Voelker, and B. Bershad, Mobisaic: An Information System for a Mobile Wireless Computing Environment, Proceedings of Workshop on Mobile Computing Systems and Applications, IEEE Press, 1994.

[10] J. E. White, *Telescript Technology: Mobile Agents*, General Magic, 1995.

[11] Y. Yokote, The Apertos Reflective Operating System: The Concept and its Implementation, Proceedings of OOPSLA'92, pp.414-434, 1992.

論文リスト

佐藤一郎, 高階モーバイルエージェントシステム, 情報処理学会プログラミング研究会報告, 3 月, 1998.

佐藤一郎, et al, “モーバイルエージェントの階層的な構成と移動” 日本ソフトウェア科学会全国大会, September, 1998. （平成１０年度日本ソフトウェア科学会高橋奨励賞受賞）

I. Satoh, “MobileSpaces: A Next Generation Mobile Agent System”, MACC’98, December, 1998.

佐藤一郎, “モーバイルエージェントの研究動向”, 日本ソフトウェア科学会チュートリアル「モーバイルエージェント」テキスト (ISSN 1341-8718-20), pp.79–109, January, 1999.

付録：ソースプログラム

MobileSpaces システムのソースファイルは、クラスファイルと同様に階層に格納されている。

```
mobilespaces -|
              |
              |- event -|
              |         |- AmbientEvent.java
              |         |- AmbientEventDispatcher.java
              |         |- AmbientEventListener.java
              |         |- AmbientEventQueue.java
              |         |- AmbientEventQueueItem.java
              |         |- AmbientQueue.java
              |         |- StatusEvent.java
              |         |- StatusListener.java
              |         |- StructureEvent.java
              |         |- StructureListener.java
              |
              |- monitor -|
              |           |- AmbientActionControl.java
              |           |- AmbientButton.java
              |           |- AmbientDestDialog.java
              |           |- AmbientDialog.java
              |           |- AmbientMenu.java
              |           |- AmbientMonitor.java
              |           |- AmbientTableControl.java
              |           |- AmbientTableEvent.java
              |           |- AmbientTableListener.java
              |           |- AmbientTableMap.java
              |           |- AmbientTableModel.java
              |           |- AmbientTableSorter.java
              |           |- AmbientTableView.java
              |           |- AmbientTreeControl.java
```

```
|           |- AmbientTreeEvent.java
|           |- AmbientTreeListener.java
|           |- AmbientTreeNode.java
|           |- AmbientTreeView.java
|           |- AmbientWindowClosing.java
|
|- protocol -|
|            |- http -|
|            |        |- AmbientHTTPContent.java
|            |        |- AmbientHTTPContent.java
|            |        |- AmbientHTTPRequest.java
|            |        |- AmbientHTTPResponse.java
|            |
|            |- matp -|
|            |        |- AmbientMATPFactory.java
|            |        |- AmbientMATPHandler.java
|            |        |- AmbientMATPRequest.java
|            |
|            |- AmbientRequest.java
|            |- AmbientRequestFactory.java
|            |- AmbientRequestParser.java
|            |- AmbientResponse.java
|            |- AmbientResponseFactory.java
|            |- AmbientResponseParser.java
|
|- system -|
|          |- Ambient.java
|          |- AmbientByteCode.java
|          |- AmbientCallMessage.java
|          |- AmbientCallback.java
|          |- AmbientCallbackManager.java
|          |- AmbientClassLoader.java
|          |- AmbientCodeBase.java
|          |- AmbientConnection.java
|          |- AmbientContainer.java
|          |- AmbientContextImpl.java
|          |- AmbientControl.java
|          |- AmbientControlImpl.java
|          |- AmbientDefaultDispatcher.java
|          |- AmbientEvent.java
```

```
|          |- AmbientEventDispatcher.java
|          |- AmbientFrame.java
|          |- AmbientFrameWrapper.java
|          |- AmbientFuture.java
|          |- AmbientIdentifier.java
|          |- AmbientInfo.java
|          |- AmbientInternalFrame.java
|          |- AmbientJFrame.java
|          |- AmbientListener.java
|          |- AmbientLoader.java
|          |- AmbientLoaderStub.java
|          |- AmbientManager.java
|          |- AmbientManagerImpl.java
|          |- AmbientMessage.java
|          |- AmbientMessageManager.java
|          |- AmbientMessageQueue.java
|          |- AmbientMethod.java
|          |- AmbientNameManager.java
|          |- AmbientObject.java
|          |- AmbientPanel.java
|          |- AmbientQueue.java
|          |- AmbientRecord.java
|          |- AmbientRuntime.java
|          |- AmbientRuntimeImpl.java
|          |- AmbientServer.java
|          |- AmbientStatus.java
|          |- AmbientThread.java
|          |- AmbientThreadGroup.java
|          |- AmbientTimeout.java
|          |- AmbientURL.java
|          |- AmbientURLConnection.java
|
|- zip -|
|       |- AmbientJar.java
|       |- AmbientManifest.java
|       |- AmbientManifestField.java
|       |- AmbientReadJar.java
|       |- AmbientReadManifest.java
|       |- AmbientWriteManifest.java
|       |- AmbientWriteJar.java
```

```
|       |- AmbientZipOutputStream.java
|       |- AmbientZipUtil.java
|
|- Context.java
|- Future.java
|- Message.java
|- NoSuchAmbientException.java
|- NoSuchCallbackException.java
|- NoSuchLocationException.java
|- NoSuchServiceException.java
|- Station.java
|- TimeoutException.java
```

5.4 MobileSpacesシステムの再コンパイル

MobileSpacesシステムに関わるすべてのソースプログラムが附属している。Java言語のJDK1.1以降に対応したJava言語コンパイラならコンパイル・実行することができる。現在、コンパイルが確認されているコンパイラを例挙する。

- Sun社 Java Development Kit (JDK) version 1.1.2, 1.1.3, 1.1.4, 1.1.5, 1.1.5, 1.1.7A, 1.1.7B
- Symantec社 Visual Cafe version 2.0, 2.5, 3.0
- Microsoft社 Visual J++ version 6.0
- Borland社 jBuilder version 2
- Metrowerks CodeWarrior

ただし、Visual J++はスクリプティング言語のユーザインターフェースを省く。