修 士 論 文

不正なホストの盗み見からモバイルエージェントを保護するセキュリティ機構の提案と実装

指導教官　渡部卓雄 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

村田真一

2001 年 2 月 15 日

要 旨

本研究の目的は，不正なホストの盗み見からモバイルエージェントを守るためのセキュリティ機構を考案し，それをアプリケーションフレームワークとして実現することにある．モバイルエージェントのセキュリティ問題は，1) 不正なエージェントが移動先のホストを攻撃する問題，2) 不正なホストがエージェントを攻撃する問題に分けることができる．既存のモバイルエージェントシステムでは，1) を考慮しているものはあるが，2) に対する有効な対処手法は確立されていない．このため，モバイルエージェントの情報が移動先のホストにより盗み見，改竄される危険性がある．

本研究ではアプリケーションとして，複数のホストを巡回して電子商取引を行う Java ベースの電子商取引エージェントを対象とする．電子商取引エージェントでは，不正なホストの盗み見から秘密情報（電子決済情報・個人情報・価格情報など）を守ることが重要である．このため，盗み見に対処する手法として，秘密情報をエージェント本体から分離してユーザのホスト上で管理し，秘密情報へのアクセスを制限するセキュリティ機構を提案する．このセキュリティ機構では，エージェントが秘密情報を持たずにネットワーク上を巡回するため，不正なホストは盗み見を行えない．しかし，エージェントは，秘密情報が必要になる度にネットワークを介してユーザのホストにアクセスする必要があり，プログラムコードが煩雑になり易い．また，セキュアなネットワーク通信やアクセス権の制御などの実装では，実装上の誤りが生じ易い．このため，本研究では，セキュリティ機構をアプリケーションフレームワークとして実現する．アプリケーションフレームワークは，電子商取引エージェントの雛型・セキュリティマネージャ・データストア・セキュリティ機能のクラスライブラリなどから構成される．本アプリケーションフレームワークを用いることにより，エージェントへのセキュリティ機構の組み込みが容易になり，エージェントのアプリケーションロジックとセキュリティポリシーに関する記述を分離できる．また，セキュリティポリシーをカスタマイズすることで，アプリケーションの処理内容や利用目的に合わせて秘密情報を保護できる．モバイルエージェントシステムには Java ベースの AgentSpace を用い，その環境上にアプリケーションフレームワークを実装した．

目 次

図 目 次

表 目 次

第 1 章

はじめに

本章では，本研究の背景と目的を述べ，最後に本論文の構成について述べる．

1.1 本研究の背景

モバイルエージェントとは，ネットワーク上のホスト間を移動し，移動先のホスト上でタスクを実行するプログラムである．ネットワーク上のホスト間を移動するプログラムとしては Java の Applet や Microsoft の Active-X が良く知られているが，これらはプログラムコードだけがネットワーク上を移動するのに対し，モバイルエージェントはプログラムの内部状態や実行状態を保持したままホスト間を移動できる点が特徴的である．この性質からモバイルエージェントは，移動前に処理していたタスクを別のホストに移動した後も継続して実行することができるため，柔軟なアプリケーションの作成が可能である．モバイルエージェントの適用分野の一つとして期待されている電子商取引では，エージェントが複数の仮想店舗を巡回し，ユーザの代わりに情報収集，電子決済，価格交渉などの処理を行うことが可能になる．しかし，モバイルエージェントを実用的なシステムで使用するためには幾つか問題があり，その中でも特に信頼性とセキュリティの問題を解決しなければならない．信頼性の問題とは，モバイルエージェントが障害により目的を達成できなくなることで，例えばホストダウンによるエージェントの消滅などがある．セキュリティの問題は，1) 不正なエージェントが移動先のホストを攻撃する問題，2) 不正なホストがエージェントを攻撃する問題に分けることができる [1]．既存のモバイルエージェントシステムでは，1) を考慮しているものはあるが，2) に対する有効な対処手法は確立されていない [2]．電子商取引をモバイルエージェントで実現するためには，2) のセキュリティ問題に対処することが重要である [3][4]．これは，決済に必要なクレジットカードの情報

や個人情報など，エージェントの秘密情報が移動先のホストによって盗み見され，悪用される危険性があるためである．本論文では，移動先のホストの盗み見からモバイルエージェントの秘密情報を保護するセキュリティ機構について述べる．

1.2 本研究の目的

本研究の目的は，不正なホストの盗み見からモバイルエージェントを守るためのセキュリティ機構を考案し，それをアプリケーションフレームワークとして実現することにある．ネットワーク上の複数のホストを巡回して電子商取引を行う Java ベースの電子商取引エージェントを対象とし，エージェントの秘密情報（電子決済情報，個人情報，価格情報など）を不正なホストの盗み見から保護するためのセキュリティ機構を提案する．エージェントにセキュリティ機構を組み込むと，プログラムコードが煩雑になり，実装上の誤りも生じやすい．このため，提案するセキュリティ機構を電子商取引エージェントのアプリケーションフレームワークとして実現する．本アプリケーションフレームワークでは，電子商取引エージェントの雛型クラスを提供する．これを基に作成される電子商取引エージェントには，不正なホストの盗み見から秘密情報を保護するためのセキュリティ機構が組み込まれ，アプリケーション毎に異なるセキュリティ要件をセキュリティポリシーとして定義できる．セキュリティポリシーは柔軟に設定および変更することが可能である．本アプリケーションフレームワークを用いて電子商取引エージェントの例題を作成し，実装したシステムを考察する．

1.3 本論文の構成

本論文の構成は以下の通りである．第２章では，本研究で扱うモバイルエージェントの概念，構成について説明し，既存のモバイルエージェントシステムを幾つか紹介する．第３章では，モバイルエージェントのセキュリティ問題とそれに対する既存の対処手法について説明し，本研究で対象とするセキュリティ問題について述べる．第４章では，本研究で提案するセキュリティ機構について述べる．第５章では，セキュリティ機構をアプリケーションフレームワークとして実現する．第６章では，アプリケーションフレームワークを用いて実際に作成した電子商取引エージェントの例題を説明し，実装したシステムを考察する．第７章では，本論文をまとめ，今後の課題について述べる．付録では，電子商取引エージェントの実行方法を説明する．

第 2 章

モバイルエージェントシステム

本章では，はじめにモバイルエージェントの基本的な概念について説明する．次に既存のモバイルエージェントシステムを幾つか紹介する．最後に，本研究で使用する AgentSpace を例に，モバイルエージェントの構造を解説する．

2.1 モバイルエージェントの概念

モバイルエージェントは比較的新しい技術であるが，既に多くのモバイルエージェントシステムが実装されている．モバイルエージェントの標準化仕様である OMG の MASIF では,「人や組織のために自律的に活動するコンピュータ・プログラム」をエージェントとし，モバイルエージェントとは「実行を開始したシステムには束縛されないエージェント」であるとしている [5][6]．しかし，この定義は全ての研究者が合意したものではなく，モバイルエージェントという用語の明確な定義は定まっていない．このため，各々の論文で定義は異なるが，一般的にモバイルエージェントは以下の性質を持ったプログラムであるとされる．

- 移動性
  ネットワーク上のホスト間を移動し，複数のホストにまたがるタスクを処理できる性質．
- 自律性
  外部から情報を取り入れ，自らの振る舞いを自分で決定できる性質．
- 協調性
  エージェント同士が通信し合い，協調的に振舞える性質．

- 適応性
  移動先のホストの環境に対応して，適切に行動できる性質.

- 局所性
  システム全体の情報を持つ必要はなく，局所的な情報のみで行動できる性質.

これらの性質のうち，自律性・協調性・適応性・局所性は，アプリケーションに依存して必要性が異なる．また，これらの性質を十分に利用した例題は少なく，モバイルエージェントにはキラーアプリケーションが無いと言われている．このため本研究では，プログラムコードと実行状態を保持したままネットワーク上を移動できる，移動性を持ったプログラムをモバイルエージェントと定義する.

図 A.2は，モバイルエージェントを用いたBtoCの電子商取引システムの例である．ユーザは自分の要求を達成できるモバイルエージェントを生成し，情報収集・価格交渉・電子決済といった仕事をモバイルエージェントに与える．モバイルエージェントは各ホストに移動し，仮想店舗と通信してユーザから与えられた仕事を遂行する.

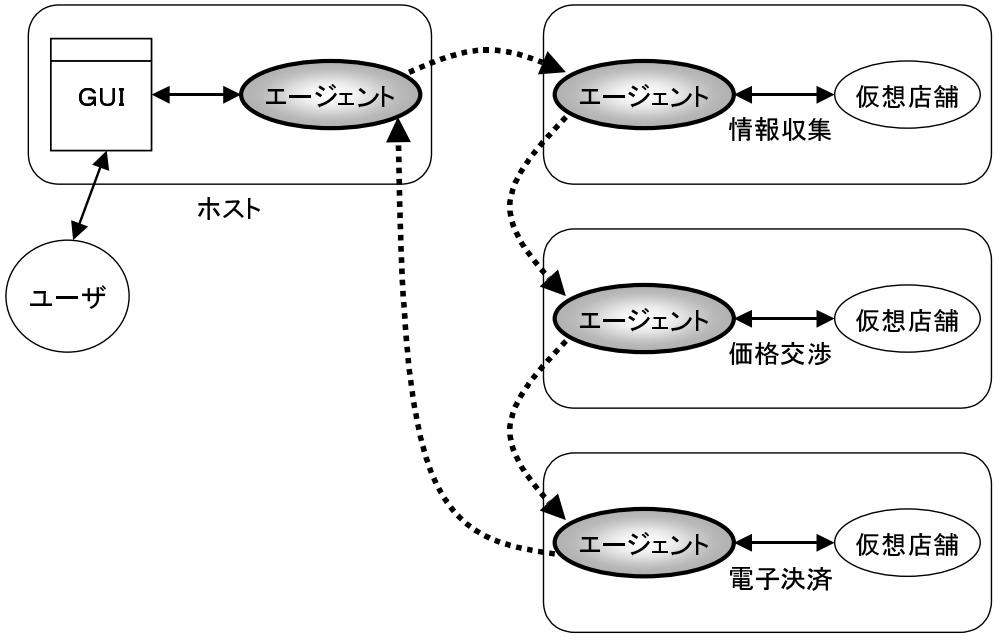

図 2.1: モバイルエージェントを用いた電子商取引の例

2.2 既存のモバイルエージェントシステム

本節では，代表的な既存のモバイルエージェントシステムを幾つか紹介する．

2.2.1 Telescript

General Magic 社により開発された世界初の商用モバイルエージェントシステムである [7]．商業的には成功しなかったが，その後のモバイルエージェント技術に大きな影響を与えた．Telescript の根本となっているアイデアは，コンピュータネットワークを「データ」だけでなく「動いているプログラム」，つまりプロセスが行き交う場にしようということである．モバイルエージェントという用語は通信技術に関するものであり，人工知能やインターフェイスエージェントのような擬人性は無いとされている．Telescript のエージェントは独自のオブジェクト指向言語である Telescript 言語で記述され，仮想機械を通じて実行される．エージェントは強い移送（Strong Migration）[8] が可能であり，プログラムコードと共に変数の値やスタックの状態などの実行環境を転送する．また，Telescript ではプレースと呼ばれる移動能力のないエージェントを初めて導入した．このプレースは各々のホストに１つ以上用意され，モバイルエージェントにリソースやサービスを提供するものである．エージェントはプレース間を移動しながら，移動先のプレースからサービスを受けたり，同一プレース上のエージェントと通信しながら処理を進める．セキュリティ対策としては，不正なエージェントから移動先のホストを守る機構として，エージェント認証，セーフティインタプリタなどの機能を備えている．

2.2.2 Aglets

IBM の東京基礎研究所が開発している Java ベースのモバイルエージェントシステムである [9]．オープンソースであり，IBM Public Licence のもとに公開されている．Aglets は Java のシリアライゼーションを利用しているため，エージェントの移動は弱い移送（Weak Migration）[8] であり，インスタンス変数は移動対象となるが，メソッド内のローカル変数やプログラムカウンタは対象外となる．エージェントは，所定のタイミングに呼び出されるコールバックメソッド群からなる Java のオブジェクトである．移動の直前，直後に呼び出されるコールバックメソッドで各種リソースの開放，獲得などを明示的に実行することができる．セキュリティ対策としては，不正なエージェントから移動先のホストを守る機構として，エージェント認証，セーフティインタプリタの機能を備えている．また，不正なホストからエージェントを保護する機構として，セキュリティドメイン認証が取り

入れられている [10]．表 2.1に Aglets のコールバックメソッドと，それぞれのメソッドが呼び出されるタイミングを示す。

メソッド名	呼び出されるタイミング
onCreation	エージェントのライフサイクルの中で１度だけ，エージェントが生成された時
onDisposing	エージェントが終了する時
onCloning	エージェントが複製される前
onClone	エージェントが複製された時に，複製により生成されたエージェントの onClone メソッドが呼び出される
onCloned	エージェントが複製された時に，オリジナルのエージェントの onCloned が呼び出される
onDispatching	移動の直前に呼び出される
onReverting	エージェントがリモートホストから呼び戻された時
onArrival	移動先のホストに到着した時
onDeactivating	エージェントが非活性化される直前
onActivation	エージェントが活性化された時

表 2.1: Aglets のコールバックメソッド

2.2.3 Voyager

ObjectSpace 社により開発されている Java ベースのモバイルエージェントシステム [11] であり，最も良く普及していると言われる．独自の ORB を基礎とし，リモートホスト上に Java オブジェクトを生成したり移動したりでき，その特別な場合としてモバイルエージェントを実現している．Java ベースのシステムであるため，Aglets と同様にエージェントの移動は弱い移送であるが，移動先で呼び出すメソッドを明示的に指定することができる．セキュリティ対策としては，リモートホスト間のエージェント通信を SSL で保護している他，不正なエージェントから移動先のホストを守る機構として，エージェント認証，セーフティインタプリタの機能を備えている．

2.2.4 Plangent

東芝 S&S 研究所により開発されているモバイルエージェントシステムである [12]．Plangent の特徴は，エージェントがプランニング機能，プラン実行機能，再プランニング機能を持つ点である．

- プランニング
  ユーザの要求を達成するための行動計画を立案する機能.
- プラン実行
  プランニングによって決定された行動計画を実行する機能.
- 再プランニング機能
  プラン実行失敗時に行動計画を生成し直す機能.

これらの機能により，エージェントに「状況」として現在のネットワーク環境や各所で提供されるサービスの状況を,「目的」としてユーザがネットワークから得たい情報や利用したいサービスの内容を与えると，エージェント自身がプランニング，プラン実行を行う自律性が実現される．

Plangent は Java により作成されたシステムであるが，エージェントはプラン記述を考慮した独自のプログラミング言語で記述される．エージェントの移動は強い移送であり，スタックやプログラムカウンタの情報も一緒に転送される．セキュリティ対策としては，不正なエージェントから移動先のホストを守る機構としてエージェント認証の他，プランニング機能に制約処理機能を追加する手法 [13] がとられている．また，不正なホストからエージェントを保護する機構としては，ホスト認証が取り入れられている．

2.2.5 AgentSpace

お茶の水女子大学の佐藤一郎氏が開発した Java ベースのモバイルエージェントシステムである [14]．エージェントは Aglets と同様に，所定のタイミングで呼び出されるコールバックメソッド群からなる．AgentSpace の特徴は，エージェントが必要とするクラスファイル全部を持って移動を行う点である．これに対し他のエージェントシステムでは，移動先のホスト上で必要になった時点でクラスが転送される．このため AgentSpace は，１度の通信で必要とする資源を転送できることからネットワークの障害に強いが，エージェントのサイズが大きくなるため移動コストが大きくなる．AgentSpace のソースコードは学生の研究用に公開されている．佐藤一郎氏はこの他に，同じく Java ベースのエージェン

トシステムである MobileSpaces も公開している．MobileSpaces は Mobile Ambients の概念を取り入れた高階モバイルエージェントシステムであり，図 2.2のように，エージェントの中にエージェントを包含することができる [15].

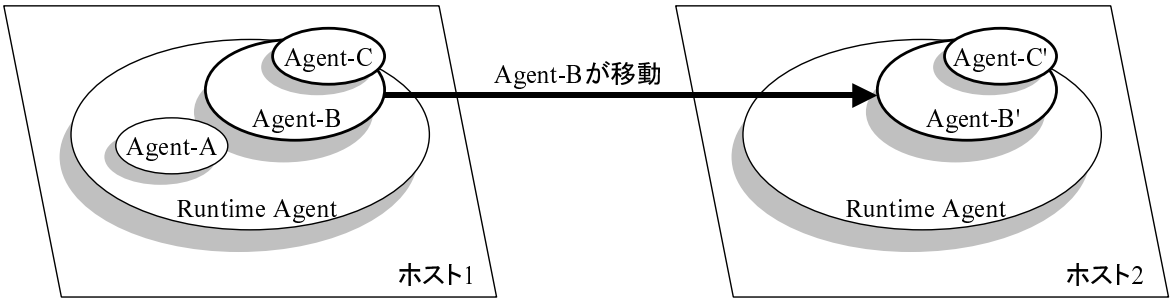

図 2.2: MobileSpaces のモバイルエージェント

2.3 モバイルエージェントの構造

ここでは，Java ベースのエージェントシステムである AgentSpace を例とし，モバイルエージェントの構造について解説する．AgentSpace は非常にシンプルなシステム構成となっている．

モバイルエージェントは，AgentSpace により生成され，AgentSpace システム上で動作する．このため，モバイルエージェントを実行する全てのホストで AgentSpace が起動していなければならない．AgentSpace システムは，モバイルエージェントの作成を可能にする API 群とランタイムシステムから構成される．ランタイムシステムは Java の仮想機械上で動作し，エージェントを管理するために以下の機能を持つ．

- エージェントの永続化
  Java のシリアライゼーション機能を利用し，エージェントを永続化する機能．永続化された情報には，エージェントのプログラムコードとインスタンス変数などの実行状態が含まれる．

- エージェントの起動
  永続化されたエージェントの情報を読み込み，エージェントを生成する機能．

- エージェントの移動
  永続化されたエージェントの情報を別のランタイムシステムに転送する機能．転送

情報を受け取ったランタイムシステムは，永続化された情報を基にエージェントを生成する．

- エージェントの停止
  エージェントプログラムの実行を停止する機能．

- エージェント間通信
  エージェント間の通信として，非同期メッセージ送信，同期メソッド呼び出し，非同期メソッド呼び出し（フューチャー通信）を行う機能．

モバイルエージェントを作成するには，エージェントの雛型クラスである Agent クラスを継承する．これにより，エージェントのコールバックメソッドをオーバーライドできる．ランタイムシステム上に生成されるモバイルエージェントは，図 2.3に示すように，生成・消滅・移動・保存・復活・複製といったライフサイクルを持つ．コールバックメソッドは，ライフサイクルにおいてエージェントの状態が変わったときに呼び出される．

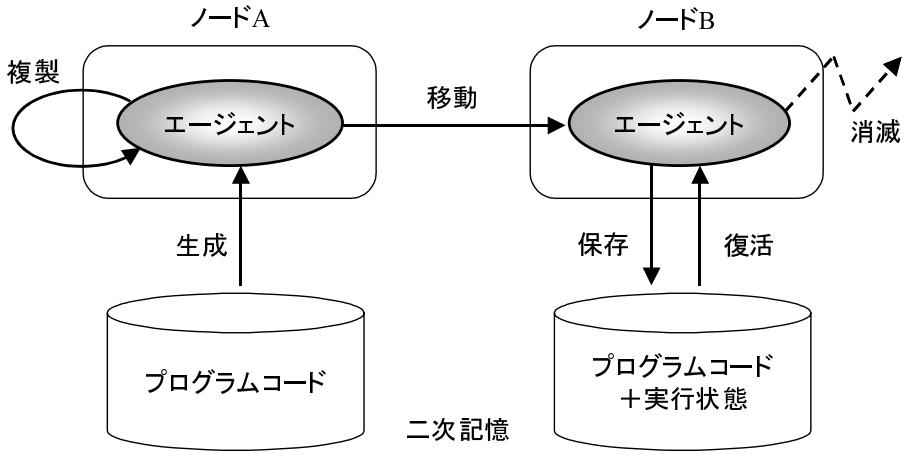

図 2.3: モバイルエージェントのライフサイクル

ランタイムシステムは，更に Agent Context，Agent Runtime から構成される．Agent Runtime は，エージェントを生成・移動・永続化．停止する機能を持ち，Agent Context は，エージェントと Agent Runtime とのインタフェースおよびエージェント間通信の機能を提供する．エージェントは，Agent Context を通じて，ホストのアドレス，他のエージェントの識別子などの実行に必要な情報を得ることができる．AgentSpace のシステム構成を図 2.4に示す．

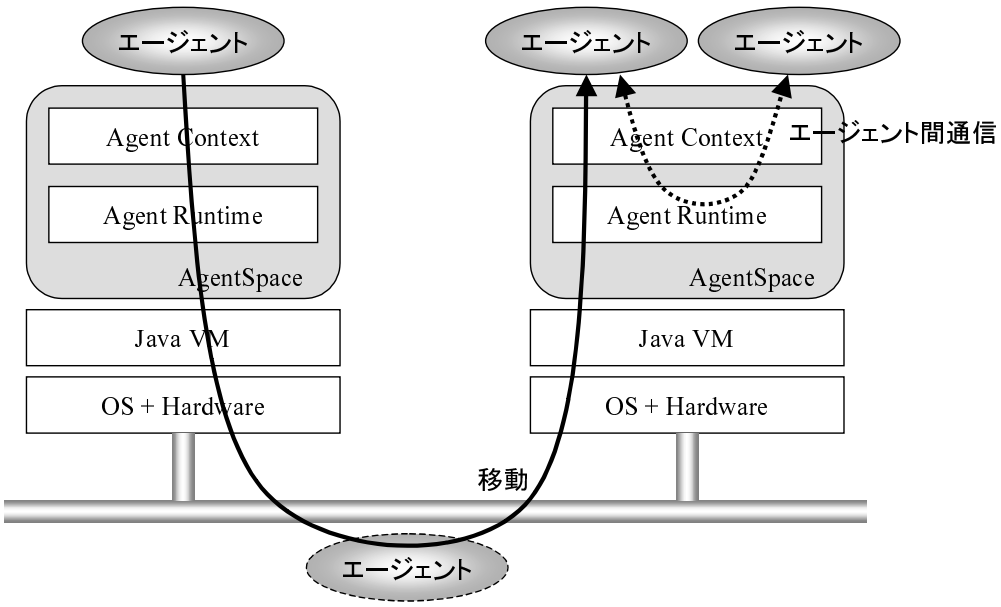

図 2.4: モバイルエージェントシステムの構成

第 3 章

想定するセキュリティ脅威

本章では，はじめに不正なエージェント，不正なホストのそれぞれのセキュリティ問題に対する既存の対処手法を紹介する．次に電子商取引システムで考慮すべきセキュリティ問題を説明し，最後に本研究で対象とするセキュリティ問題について述べる．

3.1 不正なエージェントからホストを守る手法

ネットワーク上を移動したモバイルエージェントは，移動先のホスト上でプログラムが実行される．このため，モバイルエージェントがコンピュータウィルスのように悪意を持って作られたプログラムである場合，移動先のホストのリソースが不正に使用される危険性がある．例えば，エージェントが移動先のホスト上に在るファイルを勝手に盗み見・改竄・削除・複製するようなことが考えられる．また，エージェントの作者に悪意がなくても，エージェントプログラムのバグにより，リソースが不正に使用される危険性もある．ここでは，これらのセキュリティ問題を引き起こすモバイルエージェントを不正なエージェントとする．不正なエージェントが引き起こすセキュリティ問題から移動先のホストを守る手法としては，セーフティインタプリタ，エージェント認証，PCC などが提案されており，既存のモバイルエージェントシステムでも取り入れられている．

- **セーフティインタプリタ**
  エージェントの実行をインタプリタ上に制限し，エージェントのリソースアクセスを制御する手法である．エージェントが利用可能なリソースは，移動先のホストが ACL（Access Control List）によって定義し，インタプリタがこれを読み込みリソースアクセスを制御する．既存の物では Java の Applet に見られる sandbox モデルが

この技術に相当し，悪意を持ったエージェント，エージェントプログラムのバグの両方に対応できる．

- **エージェント認証**
  エージェントプログラムの作成者およびエージェントのオーナー（エージェントオブジェクトの生成者）を認証し，信頼できるエージェントだけを実行する手法である．エージェントの作成者およびオーナーは，エージェントに電子署名を付加し，移動先のホストはこれを認証する．この他，エージェントが経由してきたホストを認証の対象にする場合もある．悪意を持ったエージェントに対応できるが，エージェントプログラムのバグには対応できず，認証を通り抜けたエージェントがリソースを不正使用しても対処できない．

- **PCC（Proof-Carrying Code）**
  PCC のキーとなるアイデアは，エージェントが安全に動作することを証明する証明書をエージェントに付加し，移動先のホストでこれをチェックすることである．証明書では，型安全やリソース使用の安全性が証明される．これにより，インタプリタやランタイムチェックなしにエージェントを安全に実行できる．

3.2 不正なホストからエージェントを守る手法

モバイルエージェントは，移動先のホストに対して情報を秘密にする事はできない．これは，移動先のホストがエージェントを実行するためには，エージェントのプログラムコード，プログラムの実行状態，インスタンス変数の値などを知らなければならないためである．このため，移動先のホストの管理者が悪意を持っている場合，エージェントが不正使用・盗み見・改竄される危険性がある．ここでは，モバイルエージェントに対してこれらの攻撃を行うホストを不正なホストとする．不正なホストの問題に対処する手法としては，ホスト認証，MobileCryptography，エージェントの難読化，N バージョン難読化エージェント，実行トレースなどの手法が提案されている．

3.2.1 ホスト認証

モバイルエージェントが移動先のホストを認証し，危険と思われるホストには近づかないことでエージェントを守る手法である．同様の保護手法として，移動先のホストが所属するセキュリティドメインを認証するセキュリティドメイン認証 [10] がある．一般にセ

キュリティドメインとは，同質のセキュリティが保持されていることが期待される領域のことで，会社の一部門の組織などが想定される．これらの手法では，エージェントは不正なホストに移動しないため攻撃を受けないが，どのように信頼できるホストおよびドメインを決定すれば良いかが問題となる．

3.2.2 Mobile Cryptography

暗号化を用いて，エージェントを盗み見，特定の目的を持った改竄から防ぐための手法である [16]．エージェントプログラムを暗号化しても，実行時に復号化を行ってから実行させるのでは平文のプログラムとなんら変わるところがない．このため MobileCryptography では，CEF（Computing with Encrypted Function）という概念で，エージェントプログラムを暗号化したまま移動先のホスト上で実行することができる．MobileCryptography は理論的なアプローチであり，現状では任意のプログラムを CEF で保護できる暗号方式は存在しない．以下に CEF プロトコルの例を示す．

- 要求
  Alice は関数 f を計算するアルゴリズムを持っている．Bob は入力 x を持っていて Alice のために $f(x)$ を計算してあげたい．しかし，Alice は Bob に f に関する本質的なことを一切教えたくない．更に，Bob と Alice は $f(x)$ の計算中に対話を必要としない．

- CEF プロトコルの手順（図 3.1参照）

  (1) Alice は f を暗号化する．

  (2) Alice は $E(f)$ を実装するプログラム $P(E(f))$ を作成する．

  (3) Alice は $P(E(f))$ を Bob に送る．

  (4) Bob は $P(E(f))$ を x に関して計算する．

  (5) Bob は $P(E(f))(x)$ を Alice に送る．

  (6) Alice は $P(E(f))(x)$ を解読して $f(x)$ を得る．

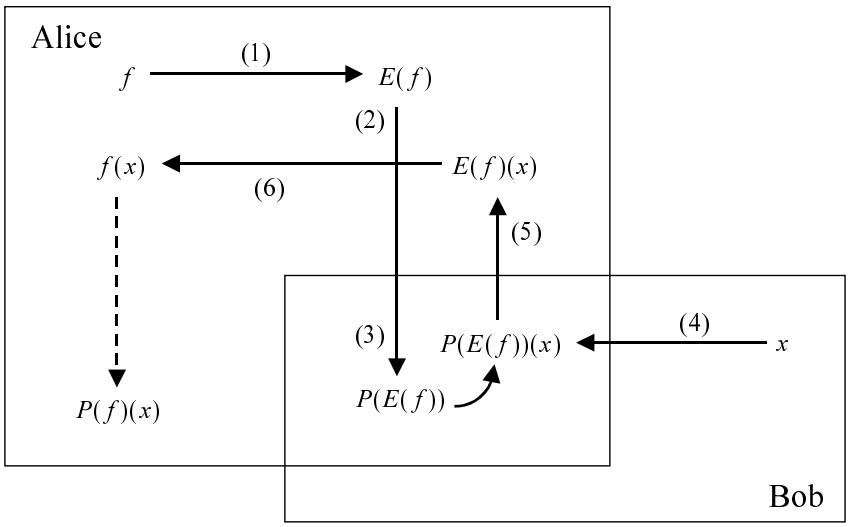

図 3.1: CEF プロトコルの例

3.2.3 エージェントの難読化

プログラムの難読化には明確な定義がある訳ではないが，プログラムのアルゴリズム解析を困難にし，解析に手間や時間を要するように仕向ける技術であると言える．エージェントを難読化することにより，不正なホストの盗み見，特定の目的を持った改竄を困難にする．現在の所，難読化には絶対的なアルゴリズムが存在しないので，モバイルエージェントのセキュリティ手法としては，難読化したエージェントプログラムを解析するのに必要な時間をエージェントのライフタイムとして制限する Time Limited Blackbox Security 手法 [3] が提案されている．以下にプログラムの難読化の簡単な例を示す．

- **意味のない名前を使用した難読化**
  図 3.2は，プログラム中の変数名や関数名に意味のない名前を使用した難読化の例である．

- **データ駆動型プログラムによる難読化**
  図 3.3は，プログラムの制御の流れを，変数 c，switch 文，ループを用いて書き換えた例である．foo2() は引数 c の初期値に外部から 1 を与えられることで foo() と同じ機能を持つ．ただし，c の初期値が 1 であることを知らない限り，foo2() が foo() と同じ機能を持つことはわからない．このため，c という変数に制御の流れを隠していると考えることができる．

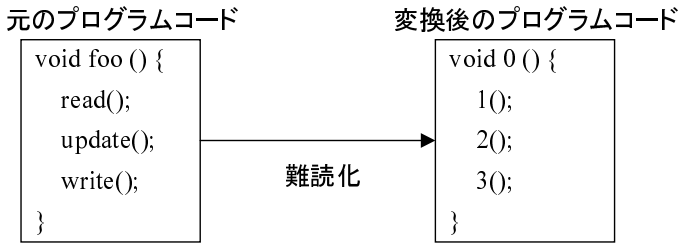

図 3.2: 意味のない名前を使用した難読化の例

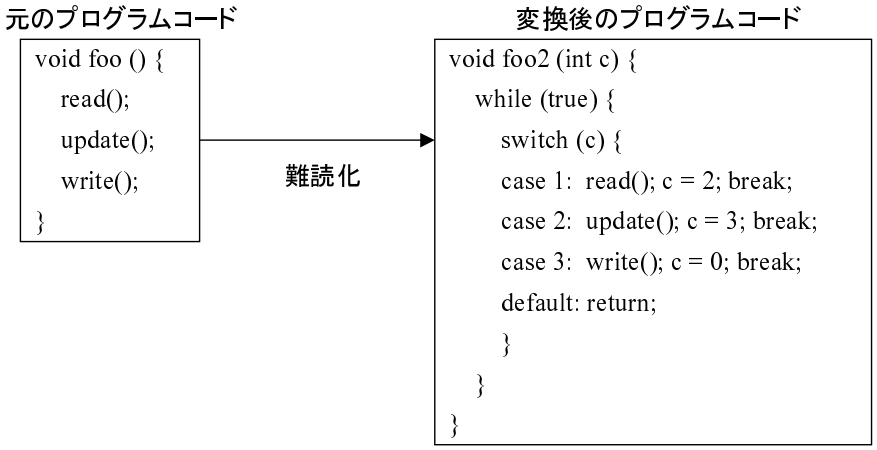

図 3.3: データ駆動型プログラムによる難読化の例

3.2.4 N バージョン難読化エージェント

エージェントの難読化を応用し，信頼性の確保とエージェントの保護を狙った手法である [17]．具体的には図 3.4に示すように，別々の難読化を施したエージェントプログラムを N 本用意して，実行時に同一のマシンもしくは異なるマシンでその N 本のエージェントを実行させ，結果の有効性を多数決で判定する．このような手法は元々ソフトウェアの信頼性向上を目的としているが，N 本のプログラムはそれぞれ難読化されており内容の解読が難しいため，不正なホストの盗み見に対処できる．また，結果の多数決を取ることで，不正なホストによるエージェントの不正利用や改竄の有無も判定できる．

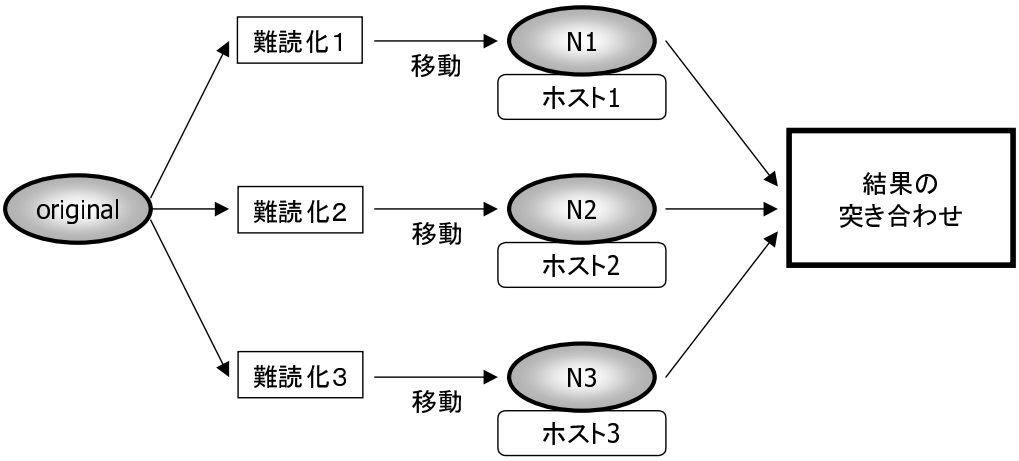

図 3.4: N バージョン難読化エージェント

3.2.5 実行トレース

エージェントの各ホスト上での実行トレースを検査することで，エージェントが改竄，不正使用されたことを検出するための手法である [4][18]．実行トレースとは，例えば図 3.5の様な，外部からエージェントへ入力された値の履歴である．実行トレースを用いた手法としては，エージェントがネットワーク上を巡回して元のホストに戻ってきてから全ての実行トレースを検査する手法や，次のホストに移動する度に前のホストの実行トレースを検査する手法が提案されている．これらの手法では，エージェントが不正使用，改竄されたことは検出できるが，攻撃そのものを防ぐことはできない．

10 read(x) 11 y=x+z 12 m=y+1 13 k=cryptInput 14 m=m+k **Code fragment**	10 x=5 11 12 13 k=2 14 **Trace of code fragment**

図 3.5: 実行トレースの例

3.3 電子商取引システムのセキュリティ問題

電子商取引という言葉は非常に広い意味で使用されるが，本研究では電子決済にクレジットカードを用いる BtoC の取引システムを対象とする．インターネット上には既に多くの電子商取引システムが存在するが，クレジットカードを用いて電子決済を行う場合に消費者が被害に遭うセキュリティ問題としては，主に以下の問題が考えられる．

1. 取引相手の仮想店舗が実在しない．
2. 仮想店舗との通信データが途中のサーバで盗聴され，悪用される．
3. クレジットカードの情報が仮想店舗により悪用される．
4. 消費者の個人情報が仮想店舗により悪用される．

これらのセキュリティ問題に対処するために，一般的に電子決済には SSL（Secure Socket Layer）プロトコルが用いられる．SSL では通信相手の認証および通信データの暗号化が行われるため，実在する取引相手と安全に通信を行う事ができる．しかし，SSL を使用したクレジットカード決済の場合，消費者から仮想店舗には安全にカード情報が転送されるものの，仮想店舗がカード情報を悪用する危険性が残る．この問題に対応する決済用のプロトコルには，Visa や MasterCard などにより開発された SET（Secure Electronic Transaction）がある．SET では，カード情報がカードの認証機関である Payment Gateway の公開鍵で暗号化されて仮想店舗に送られるため，仮想店舗が情報を盗み見することができない．以下に，インターネット上でカード決済を行う場合の一般的な購入手順を示す．

(1) 消費者が商品を見る（Web，CD-ROM）．

(2) 消費者が商品を選ぶ．

(3) 仮想店舗がオーダー・フォーム（注文票）を提供する．

(4) 消費者が仮想店舗にオーダーを送る．

(5) 消費者がクレジットカードを選択する．

(6) 消費者が仮想店舗に購入要求を送る．

(7) 仮想店舗が Payment Gateway にカードの認証依頼を送る．

(8) 仮想店舗が消費者にオーダー確認を送る．

(9) 仮想店舗が消費者に商品を発送する．

(10) 仮想店舗が Payment Gateway に支払いを要求する．

このうち SET で規定されているのは (6)(7)(8)(10) の手順である．手順 (6) で消費者から仮想店舗に送られる購入要求には，オーダー（OI：Order Information）と支払い指示（PI：Payment Instruction）が含まれる．クレジットカードの情報は PI に含まれる．OI は Payment Gateway に見せる必要はなく，PI は仮想店舗に見せる必要はない．SET では，必要のないデータを見せないようにすることでデータの悪用を防いでいる．これを実現するために購入要求のデータは図 3.6に示す内容となっている．図中の dual signature とは，OI のメッセージダイジェストと PI のメッセージダイジェストを連結したもののメッセージダイジェストである．

$$\text{dual signature} = MD(MD(OI) + MD(PI))$$

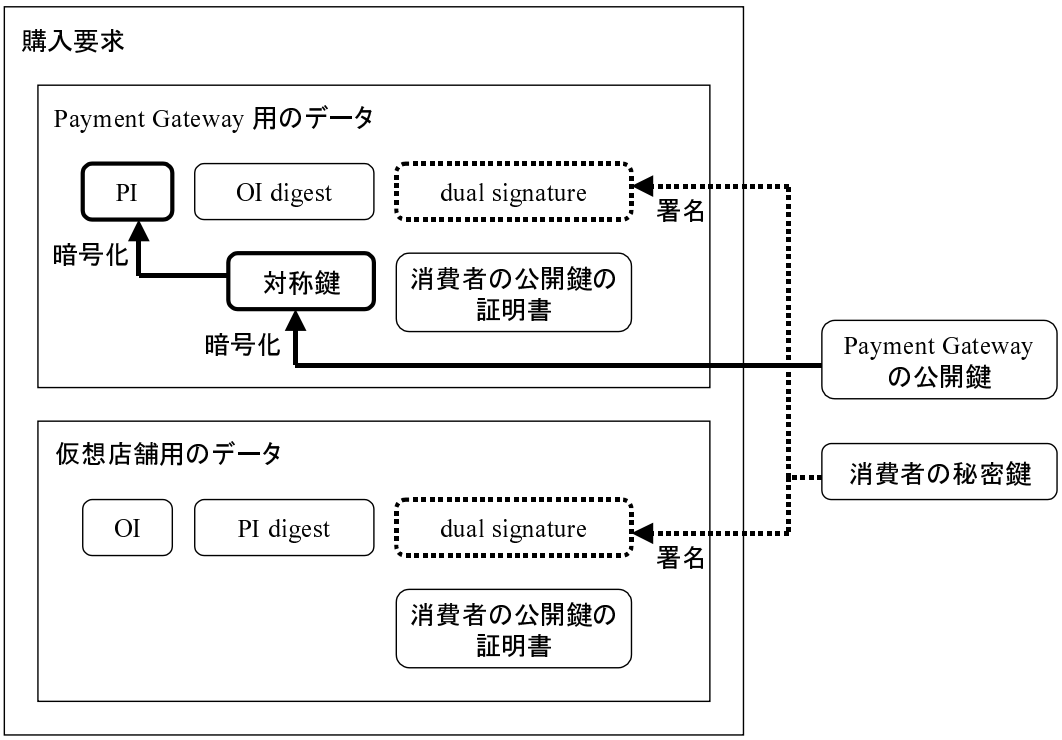

図 3.6: SET の購入要求データの内容

3.4 本研究で対象とするセキュリティ問題

本研究では，モバイルエージェントを不正なホストの盗み見から守るためのセキュリティ機構を考案する．エージェントアプリケーションには，複数のホスト上に存在する仮想店舗を巡回し，ユーザの代わりにクレジットカード決済で買物を行うモバイルエージェントを考える．本研究では，ネットワーク巡回型のこのモバイルエージェントを電子商取引エージェントと呼ぶ．電子商取引エージェントでは，モバイルエージェントのセキュリティ問題の中でも，不正なホストの盗み見に対処することが重要である．これは，決済に必要なクレジットカードの情報やユーザの個人情報が不正なホストによって盗み見され，悪用される危険性があるためである．しかし表 3.1に示すように，既存の不正なホストへの対処手法ではこの問題に十分に対処できない．

一般的に，盗み見への対処手法としては暗号化が用いられるが，モバイルエージェントの場合，エージェントの暗号化ではこれに対処できない．これは，移動先のホストがエージェントプログラムを解釈し，実行しなければならないためである．ネットワーク上を移動するエージェントが暗号化されていても，移動先のホスト上で復号化されてから実行さ

既存の対処手法	盗み見への対処
ホスト認証	事前に不正なホストがわかっていなければならない.
Mobile Cryptography	理論的には盗み見を防げるが，実行できるシステムがない.
エージェントの難読化	時間制限付きでしか盗み見に対処できない.
N バージョン難読化エージェント	時間制限付きでしか盗み見に対処できない.
実行トレース	盗み見は防げない.

表 3.1: 盗み見への対処

れるのでは，暗号化していないのと同じである．Mobile Cryptography はエージェントプログラムを暗号化したまま実行するための理論的アプローチであるが，現在はこれを実現できるシステムは存在しない．エージェントの難読化も盗み見に対処する手法であるが，現在，絶対的な難読化アルゴリズムは無く，クレジットカード情報のように有効期間の長い情報を守ることはできない．このため，電子商取引エージェントを移動先のホストの盗み見から保護することは非常に難しい問題である．本研究では，モバイルエージェント固有のセキュリティ問題のうち，エージェントの持つ秘密情報が不正なホストによって盗み見される問題を対象とする．

第 4 章

本研究のセキュリティ機構

本章では，本研究で提案するセキュリティ機構について述べる．はじめに電子商取引エージェントのネットワーク上の巡回パターンを分類し，不正なホストの盗み見に対処するために考慮すべきセキュリティ要件を整理する．次に秘密情報を盗み見から保護する手法について説明する．最後に秘密情報へのアクセスを制限するセキュリティポリシーについて，定義方法と記述例を示す．

4.1 巡回パターンの分類

電子商取引エージェントは，複数のホスト上に存在する仮想店舗を巡回し，ユーザの代わりにクレジットカード決済で買物を行う．このようなエージェントは，ネットワーク上の巡回パターンで分類することが可能である．ここでは，決済のタイミング，仮想店舗との交渉の有無をもとに，巡回後決済エージェント，巡回中決済エージェント，巡回交渉エージェントの３つに分類する．

4.1.1 巡回後決済エージェント

はじめに複数の仮想店舗を巡回して情報収集を行い，その後で決済対象とする仮想店舗と電子決済を行う電子商取引エージェントである．巡回後決済エージェントの処理手順を以下に説明し，巡回イメージを図 4.1に示す．

(1) ユーザに製品名・価格などの条件と巡回する仮想店舗のリストを指定させる．

(2) 仮想店舗を巡回し，ユーザから指定された条件に合う商品の価格リストなどの情報を収集する．

(3) ユーザのホストに戻る．

(4) 収集した情報を分析し，決済対象の仮想店舗を決定する．または，収集した情報をユーザに提示し，決済対象の仮想店舗を指定させる．

(5) 決済対象の仮想店舗に移動し，決済を行う．

(6) ユーザのホストに戻り，取引結果を報告する．

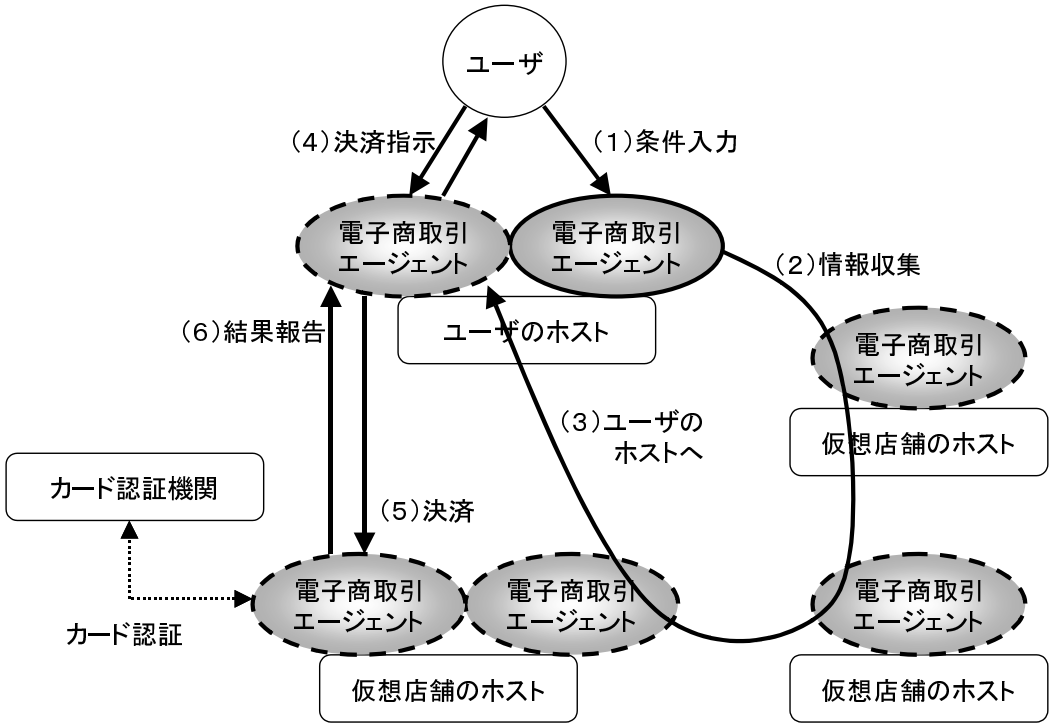

図 4.1: 巡回後決済エージェント

巡回後決済エージェントは，複数の仮想店舗から情報を収集する時に，仮想店舗から得た商品の価格リストを持ったまま別の仮想店舗に移動する．このため，不正な仮想店舗のホストが，エージェントが持っている他店舗の情報を盗み見する危険性がある．

4.1.2 巡回中決済エージェント

仮想店舗の巡回中に条件に合う店舗が見つかれば，その時点で電子決済を行う電子商取引エージェントである．巡回中決済エージェントの処理手順を以下に説明し，巡回イメージを図 4.2に示す．

(1) ユーザに製品名・価格などの条件と巡回する仮想店舗のリストを指定させる．

(2) 仮想店舗へ移動し，ユーザから指定された条件に合う商品の価格などの情報を獲得する．

(3) 獲得した情報が条件に合えば決済を行う．例えば，店舗の商品価格がユーザから指定された価格よりも安ければ決済を行う．条件に合わなければ別の仮想店舗へ移動する．

(4) ユーザのホストに戻り，取引結果を報告する．

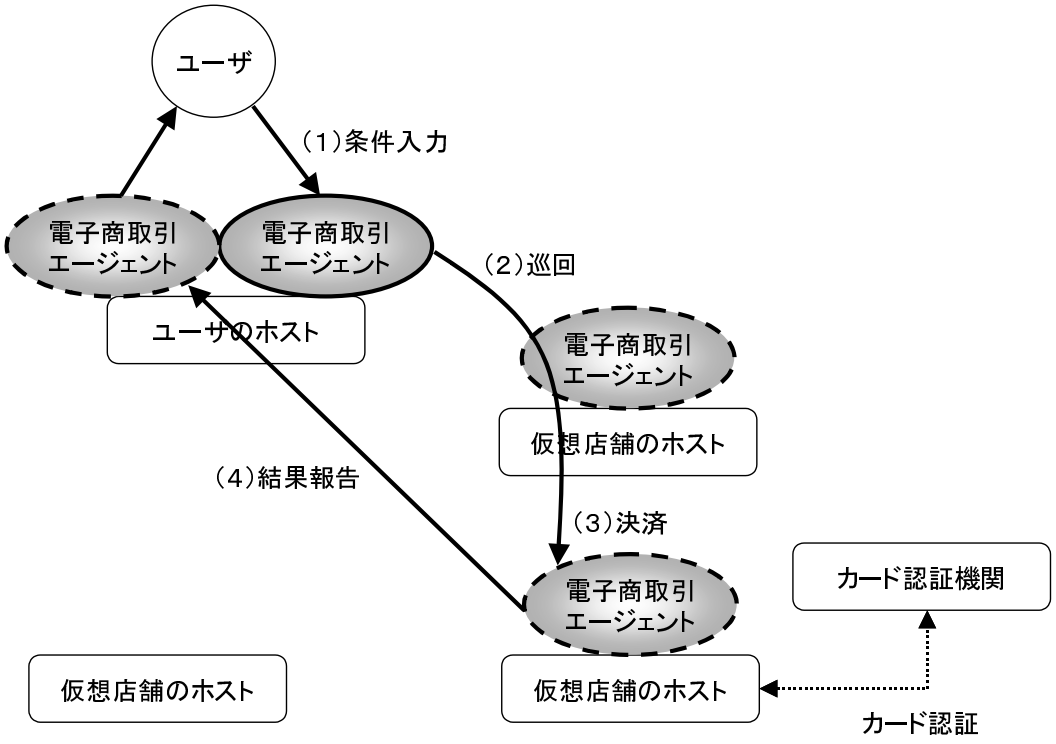

図 4.2: 巡回中決済エージェント

巡回中決済エージェントは，移動先のホスト上で決済対象とする仮想店舗を決定するため，決済対象を決定するための基準となる情報が不正な仮想店舗のホストにより盗み見される危険性がある．また，巡回中に決済を行うため，エージェントが持つ決済情報や個人情報なども盗み見の対象となる．

4.1.3 巡回交渉エージェント

はじめに複数の仮想店舗を巡回して情報収集を行い，その後で仮想店舗と価格交渉などの交渉を行う電子商取引エージェントである．仮想店舗との間でどのように交渉を進めるかは，実装する交渉手順によって異なる．巡回交渉エージェントの処理手順を以下に説明し，巡回イメージを図 4.3に示す．

(1) ユーザに製品名・価格などの条件と巡回する仮想店舗のリストを指定させる．

(2) 仮想店舗を巡回し，ユーザから指定された条件に合う商品の価格リストなどの情報を収集する．

(3) ユーザのホストに戻る．

(4) 収集した情報を分析し，交渉対象の仮想店舗を決定する．または，収集した情報をユーザに提示し，交渉対象の仮想店舗を指定させる．

(5) 交渉対象の仮想店舗に移動し，交渉を行う．

(6) 交渉が成立すれば，移動先の仮想店舗と決済を行う．交渉が成立しなければ，別の仮想店舗に移動して交渉や決済を行う．

(7) ユーザのホストに戻り，交渉結果および取引結果を報告する．

巡回交渉エージェントは，複数の仮想店舗から情報収集を行うため，巡回後決済エージェントと同様に，エージェントが持っている他店舗の情報が盗み見される危険性がある．また，情報収集後は複数の仮想店舗を巡回して交渉や決済を行うため，交渉や決済に必要な情報も盗み見の対象となる．

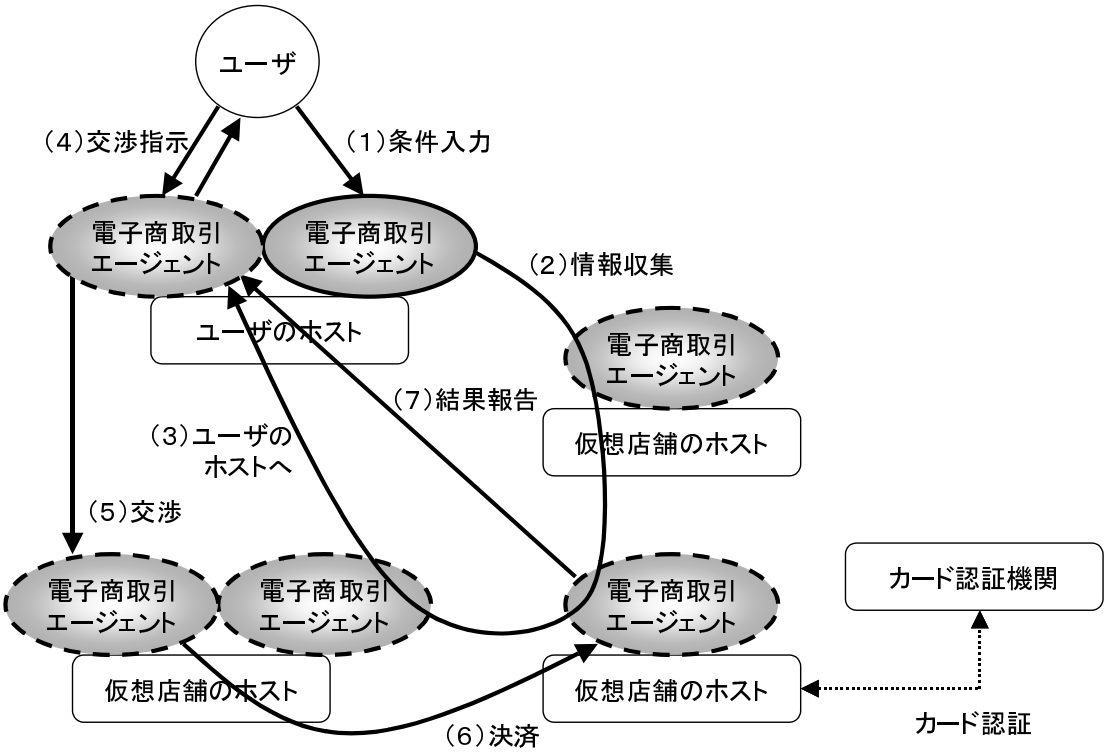

図 4.3: 巡回交渉エージェント

4.2 セキュリティ要件

分類した３つの巡回パターンでネットワーク上を巡回する電子商取引エージェントは，それぞれのエージェント毎に盗み見の対象となる情報が異なる．本研究では，それぞれの電子商取引エージェントにおいて，盗み見から守らなければならない情報をエージェントの秘密情報と呼ぶ．秘密情報を不正なホストの盗み見から保護するには，セキュリティ要件として，情報の公開先，情報の保護手法を考慮しなければならない．これら２つのセキュリティ要件の組み合わせは，秘密情報毎に考慮すべきものが異なる．また，エージェントの処理内容や利用方法にも依存する．このため，本セキュリティ機構では，適用するセキュリティ要件をセキュリティポリシーとして定義することで，各々のエージェントでセキュリティ要件をカスタマイズできるようにしている．

4.2.1 情報の公開先

情報の公開先とは，エージェントの秘密情報をネットワーク上のどのホストに対して公開するのかという事である．公開先に指定するホストだけが秘密情報を見ることができる．例えば，ユーザがある店舗の会員となっている場合，会員情報は，その会員となっている店舗だけに教えたい情報である．また，電子決済に必要なクレジットカードの情報は，カードの認証機関だけに教えたい情報である．本研究では，電子商取引エージェントが保持する情報を，公開先によって以下のように分類する．

- ネットワーク上の全てのホストに公開する情報
- 特定の仮想店舗のあるホストにだけ公開する情報
- 特定のカード認証機関のあるホストにだけ公開する情報
- エージェントを作成したユーザのホスト以外は，全てのホストに対して秘密にする情報

4.2.2 情報の保護手法

情報の保護手法とは，エージェントの秘密情報を公開先で指定される特定のホストだけに公開し，その他のホストには秘密にするための手法である．本セキュリティ機構では，既存の暗号アルゴリズムおよび暗号プロトコルを保護手法として利用する．一般的に，情報は SSL プロトコルを用いて保護すれば安全である．しかし，クレジットカードによる電子決済では，不正な仮想店舗のホストからカード情報を守るために，カード情報を SET プロトコルによって保護しなければならない．本セキュリティ機構では，セキュリティポリシーにより，秘密情報毎に適用する保護手法を選択できる．また，独自の暗号プロトコルを追加することも可能である．

4.3 秘密情報の保護機構

本セキュリティ機構では，図 4.4に示すように，電子商取引エージェント本体とエージェントの秘密情報を分離することで，不正なホストの盗み見から秘密情報を保護する．ユーザのホスト上で秘密情報を管理するモバイルエージェントを秘密情報管理エージェントと呼ぶ．また，秘密情報は持たないが電子商取引エージェントのアプリケーションロジックを持ち，ネットワーク上を巡回するモバイルエージェントを巡回エージェントと呼ぶ．巡

回エージェントと秘密情報管理エージェントは異なるホスト上で動作するが，ネットワークを介して連携することで１つの電子商取引エージェントとして機能する．

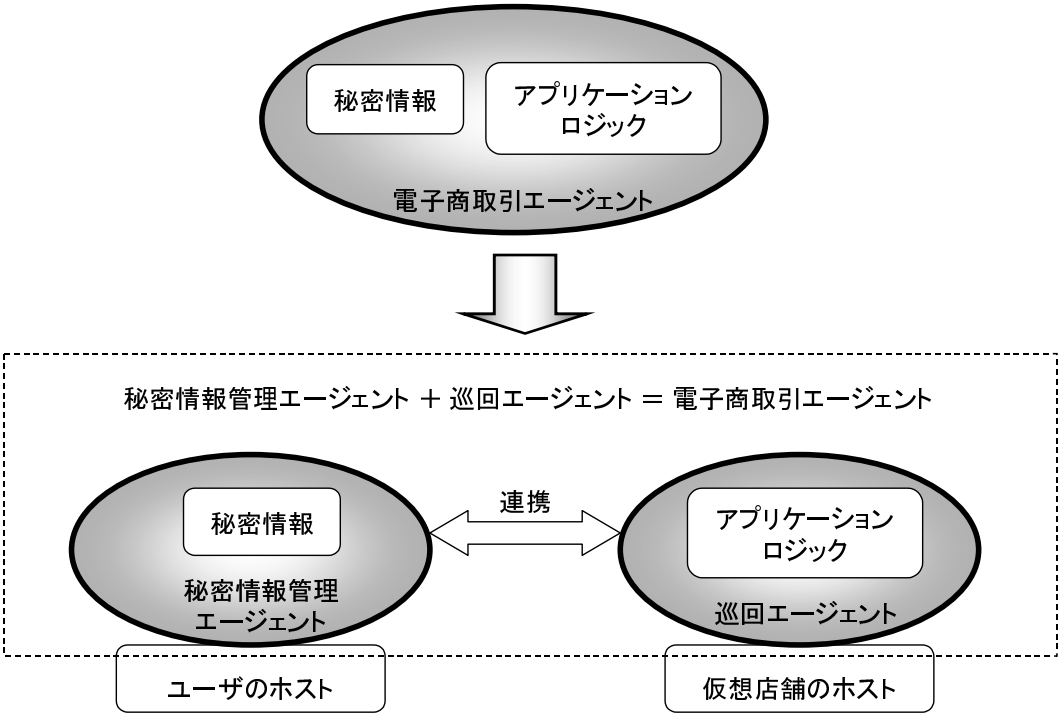

図 4.4: 電子商取引エージェントの分離

この機構では，巡回エージェントが秘密情報を持たずにネットワーク上を巡回するため，不正な仮想店舗のホストは秘密情報を盗み見できない．しかし，巡回エージェントは，仮想店舗と決済などを行うタイミングで秘密情報が必要になる．このため図 4.5のように，巡回エージェントは必要になった時点で秘密情報管理エージェントに要求を送り，暗号化などの適切な処理を施した秘密情報を受け取る．ネットワーク上の全ての巡回エージェントが秘密情報管理エージェントに要求を送ることができるが，秘密情報管理エージェントはセキュリティポリシーで許可される巡回エージェントだけに秘密情報を返す．本セキュリティ機構では，セキュリティマネージャおよびデータストアの機能を用い，秘密情報へのアクセス制限およびエージェント間のセキュアな通信を実現する．

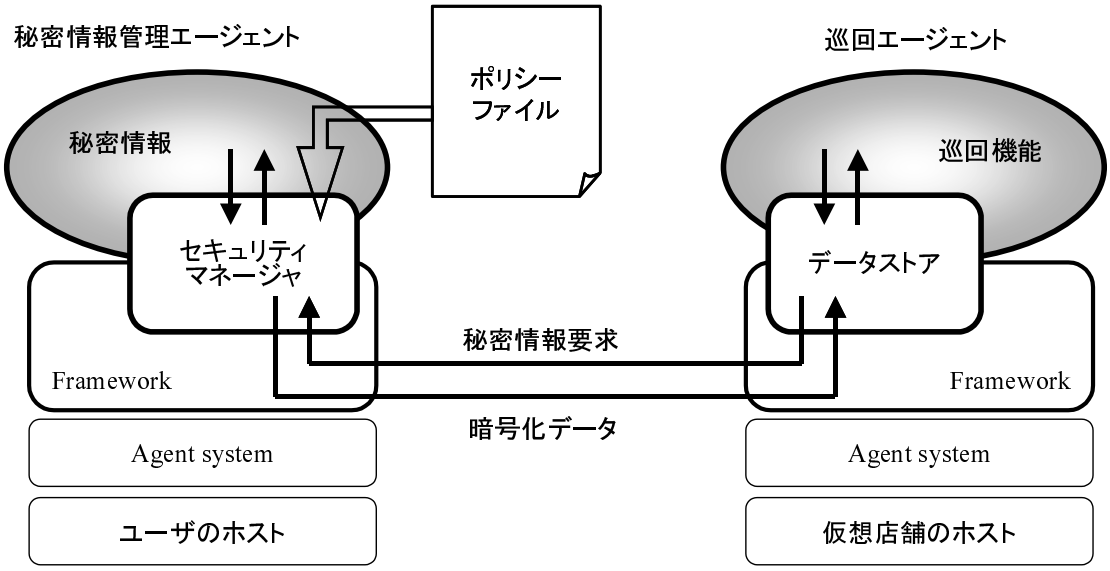

図 4.5: 巡回エージェントと秘密情報管理エージェントの連携

4.3.1 巡回エージェント

電子商取引エージェントのアプリケーションロジックを持ち，ネットワーク上の複数の仮想店舗を巡回してユーザの代わりに電子商取引を行うエージェントである．巡回エージェントが持つ情報は，不正な仮想店舗のホストによって盗み見される危険性があるため，秘密情報を持つことはできない．このため，巡回エージェントは，決済時などの秘密情報が必要になった時点で秘密情報管理エージェントに要求を送り，暗号化などの適切な保護手法で保護された秘密情報を受け取る．巡回パターンによっては仮想店舗で獲得した情報も盗み見の対象となるが，以下の手法で保護することができる．

- 別の仮想店舗へ移動する前に秘密情報管理エージェントへ送信する．
- 別の仮想店舗へ移動する前にユーザの公開鍵で暗号化する．

4.3.2 秘密情報管理エージェント

電子商取引エージェントの秘密情報を管理するエージェントである．本研究では，ユーザのホストを安全なホストと考えるため，秘密情報管理エージェントはユーザのホストに留まり，他のホストには移動しない．巡回エージェントからの要求を受け取り，暗号化などの適切な保護手法で保護した秘密情報を返す．秘密情報へのアクセス制御は，セキュリ

ティマネージャを用いて実現する．

4.3.3 セキュリティマネージャ

セキュリティポリシーに定義されている秘密情報へのアクセス権および保護手法を読み込んで制御する機能であり，秘密情報管理エージェントに組み込まれる．巡回エージェントから秘密情報管理エージェントへの要求はまずセキュリティマネージャが受け取り，要求がセキュリティポリシーに違反しないことを監視する．巡回エージェントが秘密情報へのアクセスを許可されていれば，秘密情報管理エージェントから秘密情報を取り出し，セキュリティポリシーに定義されている保護手法を適用して巡回エージェントへ送る．

4.3.4 データストア

巡回エージェントに組み込まれ，秘密情報管理エージェントとのセキュアなリモート通信や一度受け取った秘密情報のキャッシュを行う．巡回エージェントは，秘密情報が必要になる度に秘密情報管理エージェントへ要求を送る必要があるが，データストアの機能を用いることで，ネットワーク通信を意識することなく秘密情報を保護できる．

4.4 セキュリティポリシーの導入

秘密情報に適用するセキュリティ要件は，セキュリティポリシーとして外部のポリシーファイルに定義する．これにより，セキュリティポリシーに関する記述をエージェントのアプリケーションロジックと分離する．ポリシーファイルには，秘密情報の識別子と，その情報に適用するセキュリティポリシーの組み合わせを記述する．秘密情報毎にセキュリティポリシーを定義することで，ポリシーの柔軟な設定が可能である．また，ポリシーファイルの内容を変更することで，エージェントの処理内容や利用方法に合わせてセキュリティポリシーをカスタマイズできる．ポリシーファイルでは，以下の項目を設定する．

- **set**
  巡回エージェントと秘密情報管理エージェントの間で SET プロトコルを用いて通信を行う場合に必要な，SET プロトコル固有の情報である．カードの認証機関である Payment Gateway の公開鍵で暗号化しなければならないデータの識別子を記述する．

- **data**
  巡回エージェントが秘密情報管理エージェントの管理しているデータにアクセスするためのセキュリティポリシーであり，次の情報を記述する．
  - データの識別子
  - データの保護手法
  - データへのアクセス権限があるホスト
  - 仮想店舗を識別するための情報
  - 巡回エージェントの状態（ユーザのホスト上，巡回中，決済中，交渉中）
  - アクセスの種類（書き込み，読み込み）
- **method**
  巡回エージェントが秘密情報管理エージェントのメソッドを呼び出す場合のセキュリティポリシーであり，次の情報を記述する．
  - メソッドの識別子
  - メソッドの復帰値の保護手法
  - メソッド呼び出し権限があるホスト
  - 仮想店舗を識別するための情報
  - 巡回エージェントの状態（ユーザのホスト上，巡回中，決済中，交渉中）
  - アクセスの種類（呼び出し）

これらの項目からなるセキュリティポリシーは，外部のポリシーファイルに XML を用いて定義する．ポリシーの記述に XML を用いることで，階層化されたポリシーの内容を容易に記述できる．また，XML はテキスト形式のデータであるため，各ユーザがポリシーをカスタマイズするために特別なエディタを必要としない．セキュリティ機構の実装においても既存のパーサーを利用できる．以下に，XML でセキュリティポリシーを定義するための DTD およびポリシーファイルの記述例を示す．

- **ポリシーファイルの DTD**

```
<!-- ポリシーは set，data，method の項目を持つ -->
<!ELEMENT policy (set?, data*, method*)>
<!-- SET プロトコル固有の情報 -->
<!ELEMENT set (payment-gateway)*>
<!ELEMENT payment-gateway (#PCDATA)>
```

```
<!-- データにアクセスするための情報 -->
<!ELEMENT data (data-name+, protect, host+)>
<!ELEMENT data-name (#PCDATA)>
<!-- メソッドを呼び出すための情報 -->
<!ELEMENT method (method-name+, protect, host+)>
<!ELEMENT method-name (#PCDATA)>
<!-- データアクセス，メソッド呼び出しに共通の情報 -->
<!ELEMENT protect EMPTY>
<!ELEMENT host (agent)+>
<!ELEMENT agent (phase)+>
<!ELEMENT phase (access)+>
<!ELEMENT access EMPTY>
<!ATTLIST protect name (unguarded | ssl | set) #REQUIRED>
<!ATTLIST host name CDATA #REQUIRED>
<!ATTLIST agent name CDATA #REQUIRED>
<!ATTLIST phase name (any | owner | itinerant |
                      purchase | negotiate) #REQUIRED>
<!ATTLIST access name (dataset | dataget | call) #REQUIRED>
```

- **ポリシーファイルの例**

  この例は，次の２つのポリシーの内容を定義したものである．

  – データアクセス

    巡回エージェントがクレジットカードで電子決済を行う場合に，秘密情報管理エージェントの決済情報にアクセスするためのポリシーである．アクセスが許可される巡回エージェントは，IP アドレスが 192.168.1.2 または 192.168.1.3 のホスト上に存在し，”CN=shop1, ...”または”CN=shop2, ...”で識別される仮想店舗と決済を行う状態でなければならない．このような巡回エージェントに対し，決済情報の読み込みだけが許可される．決済情報は SET プロトコルを用いて保護される．

  – メソッド呼び出し

    巡回エージェントが秘密情報管理エージェントの foo メソッドを呼び出す場合のポリシーである．巡回エージェントがユーザのホスト（192.168.1.1）上に存在する場合に，foo メソッドの呼び出しが許可される．

```
<policy>
  <set> <!-- SET 決済用ポリシー -->
        <!-- カードの番号・有効期限を Payment Gateway の公開鍵で保護する -->
    <payment-gateway>CARD_NO</payment-gateway>
    <payment-gateway>CARD_TERM</payment-gateway>
  </set>
  <data> <!-- データアクセスポリシー -->
         <!-- ユーザの名前，郵便番号，住所，電話番号，年齢， -->
         <!-- カードの番号・有効期限，支払い回数を保護する  -->
    <data-name>NAME</data-name>
    <data-name>ZIPCODE</data-name>
```

```
    <data-name>ADDRESS</data-name>
    <data-name>PHONE</data-name>
    <data-name>AGE</data-name>
    <data-name>CARD_NO</data-name>
    <data-name>CARD_TERM</data-name>
    <data-name>INSTALLMENT</data-name>
    <protect name="set" />
    <host name="192.168.1.2">
      <agent name="CN=shop1, OU=Watanabe lab, O=Jaist,  ... ">
        <phase name="purchase">
          <access name="dataget" />
        </phase>
      </agent>
    </host>
    <host name="192.168.1.3">
      <agent name="CN=shop2, OU=Watanabe lab, O=Jaist,  ... ">
        <phase name="purchase">
          <access name="dataget" />
        </phase>
      </agent>
    </host>
  </data>
  <method> <!-- メソッドアクセスポリシー -->
           <!-- foo メソッドの呼び出しを制限する -->
    <method-name>foo</method-name>
    <protect name="unguarded" />
    <host name="192.168.1.1">
      <agent name="CN=customer, OU=Watanabe lab, O=Jaist,  ... ">
        <phase name="owner">
          <access name="call" />
        </phase>
      </agent>
    </host>
  </method>
</policy>
```

第 5 章

アプリケーションフレームワークとその実装

本章では，本研究で実装したアプリケーションフレームワークについて説明する．アプリケーションフレームワークは，４章で提案したセキュリティ機構を Java で実装したものである．実装のベースとなるモバイルエージェントシステムには AgentSpace（1999'4'30 版）を使用した．また，Java の拡張機能として Sun 社が提供する JCE（Java Cryptography Extension）1.2.1，JSSE（Java Secure Socket Extension）1.0.2，Java Project X Technology Release 2 を使用した．

5.1　アプリケーションフレームワークの構成

アプリケーションフレームワークは，電子商取引エージェントの雛型，セキュリティマネージャ，データストア，セキュリティ機能のクラスライブラリなどから構成される．各々の説明を以下に述べ，それらのフレームワーク内での関係を図 5.1に示す．

- **電子商取引エージェントの雛型**
  巡回エージェントおよび秘密情報管理エージェントの雛型クラスである．電子商取引エージェントのアプリケーションを作成する場合は，これらのクラスを継承することで容易に秘密情報をエージェント本体から分離できる．巡回エージェントは，データストア機能の実装クラス，巡回パターンの実装クラスを持つ．巡回パターンクラスには，巡回後決済エージェント・巡回中決済エージェント・巡回交渉エージェントの各々の実装クラスがあり，これらを選択して組み込むことができる．

- **セキュリティマネージャ**
  本セキュリティ機構におけるセキュリティマネージャの機能を実装したクラスである．秘密情報管理エージェントに組み込まれ，外部のポリシーファイルに XML で定義されているセキュリティポリシーを読み込み，秘密情報へのアクセスを制限する．

- **データストア**
  本セキュリティ機構におけるデータストアの機能を実装したクラスである．巡回エージェントに組み込まれ，巡回エージェントが秘密情報管理エージェントにアクセスするための機能を持つ．

- **セキュリティ機能のクラスライブラリ**
  JCE および JSSE の機能をラップし，既存の暗号アルゴリズム，暗号プロトコル，電子署名などの基本的なセキュリティ機能を提供するクラス群である．電子商取引エージェントの雛型やセキュリティマネージャから使用されている．フレームワークを用いてアプリケーションを作成する場合にも使用できる．

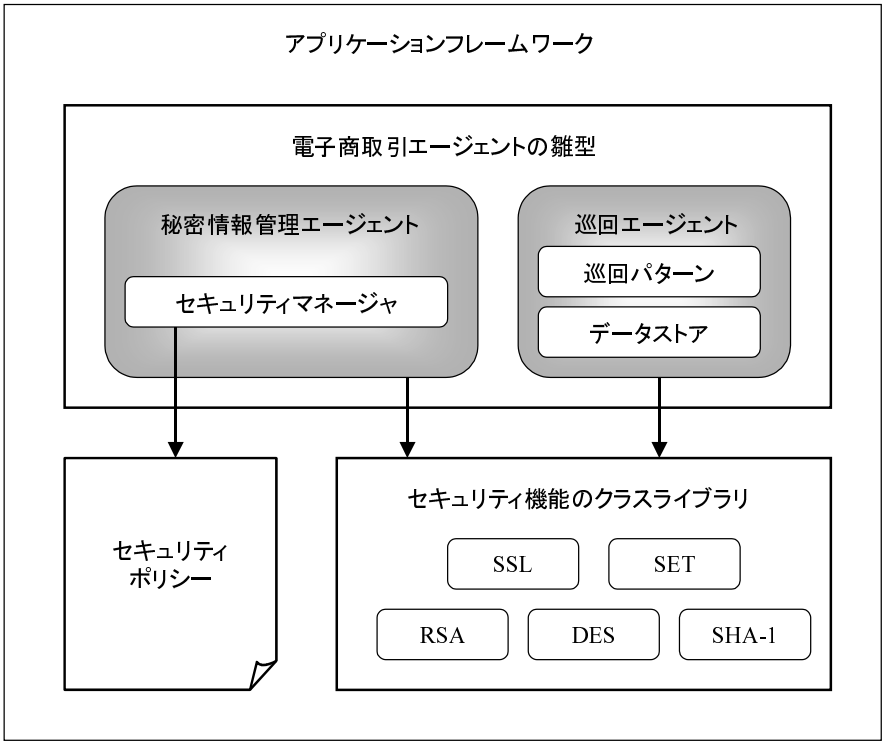

図 5.1: アプリケーションフレームワークの構成

5.1.1 パッケージ構成

アプリケーションフレームワークは Java のクラスファイル群から構成される．図 5.2 に，これらのクラスファイルのパッケージ構成を示す．

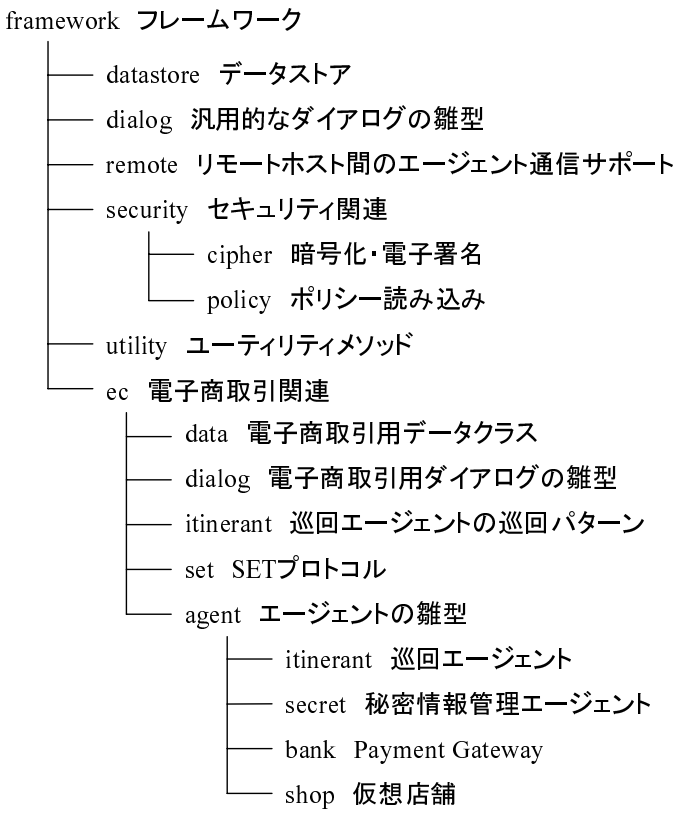

図 5.2: アプリケーションフレームワークのパッケージ構成

5.1.2 クラス構成

アプリケーションフレームワークを構成する主なクラスの構成を図 5.3に示す．図は UML のクラス図であるが，クラスの属性や操作を省略してクラス名だけを記述したものである．表 5.1で図中の各々のクラスについて説明する．

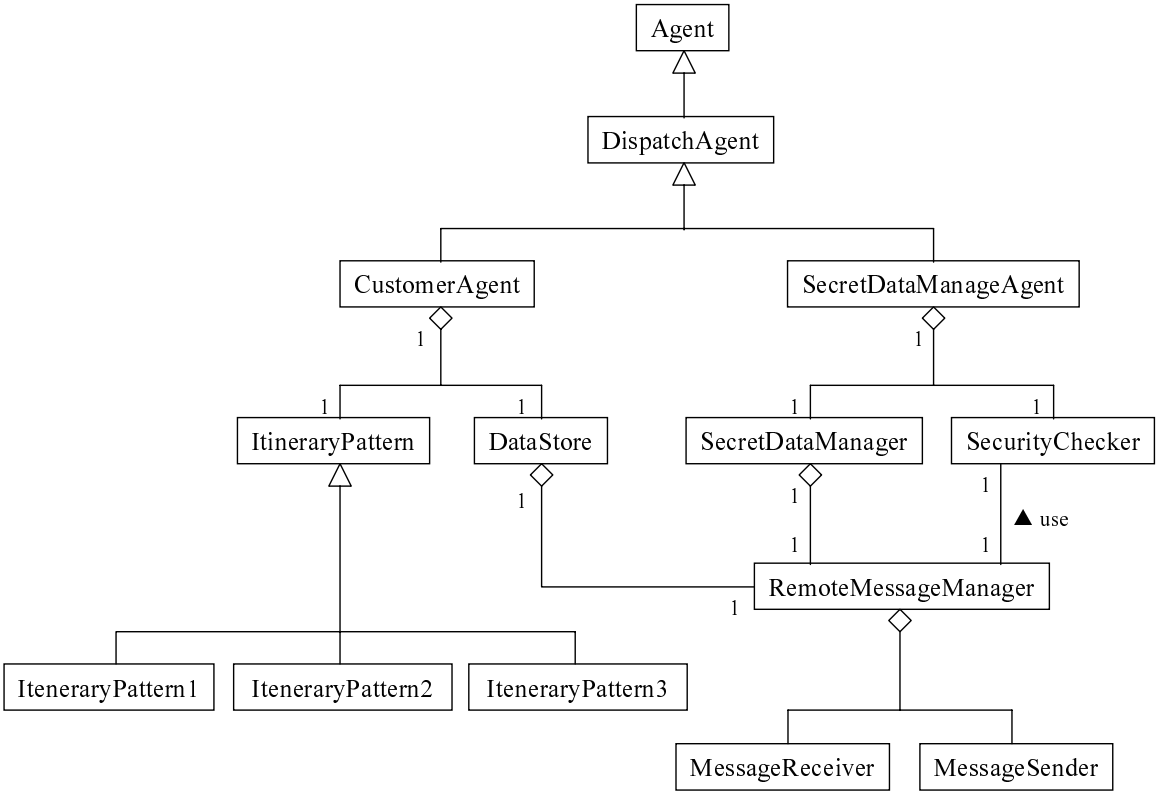

図 5.3: アプリケーションフレームワークのクラス構成

クラス名	機能
Agent	AgentSpace で提供されるエージェントの雛型クラス.
DispatchAgent	ネットワーク上の自動巡回機能を持つエージェントの雛型クラス. リモートホスト間通信への対応機能も持つ.
CustomerAgent	巡回エージェントの雛型クラス
SecretDataManageAgent	秘密情報管理エージェントの雛型クラス
ItineraryPattern	巡回パターンの雛型クラス.
ItineraryPattern1	巡回後決済エージェントの実装クラス.
ItineraryPattern2	巡回中決済エージェントの実装クラス.
ItineraryPattern2	巡回交渉エージェントの実装クラス.
DataStore	データストア機能の実装クラス.
SecretDataManager	秘密情報を管理するクラス.
SecurityChecker	セキュリティマネージャ機能の実装クラス.
RemoteMessageManager	エージェント間で, セキュアなリモート通信を実現するためのクラス.
MessageRceiver	リモート通信メッセージの受信機能の雛型クラス.
MessageSender	リモート通信メッセージの送信機能の雛型クラス.

表 5.1: アプリケーションフレームワークの主なクラスの機能

5.2 AgentSpace システムの変更

アプリケーションフレームワークの実装は AgentSpace の環境上で行ったが，幾つか AgentSpace 自体の機能を拡張した．ここでは，本研究で拡張した機能について説明する．

- **JDK1.3 への対応**
  本研究では，Java の実行環境として JDK1.3 を使用した．使用したバージョンの AgentSpace は，JDK1.1 系の環境では動作するが JDK1.3 の環境には対応していない．このため，AgentSpace のクラスローダー関連の機能を修正して JDK1.3 に対応した．

- **コールバックメソッドの追加**
  AgentSpace で作成されるモバイルエージェントは，エージェントの雛型クラスである Agent クラスを継承しなければならない．この Agent クラスは，表 5.2に示すコールバックメソッドを持つ．これらのコールバックメソッドは，エージェントの実行状態が変化する時にランタイムシステムから呼び出される物である．フレームワークでは，表 5.3に示す isMoveable と getName のコールバックメソッドを Agent クラスに追加した．エージェントは，isMoveable メソッドの復帰値により，移動を許可するかどうかを制御できる．

- **エージェントのパッケージ化**
  使用したバージョンの AgentSpace は，パッケージ化されたエージェントクラスを扱うことができない．これは，AgentSpace で提供されている MakeAgent コマンドがパッケージに対応していないためである．MakeAgent コマンドは，Java のクラスファイルからモバイルエージェントのアーカイブを作成するコマンドある．本研究では，MakeAgent コマンドを修正してパッケージ対応を可能にした．

メソッド名	呼び出されるタイミング
void init()	クラスファイルからエージェントのアーカイブを作成する時に一度だけ呼び出される.
void create()	エージェントが作成される時に一度だけ呼び出される.
void destroy()	エージェントが消滅する直前に呼び出される.
void dispatch(URL)	エージェントが他のホストに移動する時に，移動の直前に呼び出される．引数は，移動先のホストの URL である.
void arrive()	エージェントが移動先のホストに到着した時に呼び出される.
void suspend()	エージェントが永続化される直前に呼び出される.
void resume()	永続化されたエージェントが，再び動作可能になった時に呼び出される.
void duplicate()	エージェントが複製される直前に呼び出される.
void parent(AgentIdentifier)	エージェントが複製された時に，複製の元となったエージェントで呼び出される．引数は，複製により生成されたエージェントの識別子である.
void child(AgentIdentifier)	エージェントが複製された時に，複製により生成されたエージェントで呼び出される．引数は，複製の元となったエージェントの識別子である.

表 5.2: AgentSpace のコールバックメソッド

メソッド名	呼び出されるタイミング	復帰値
boolean isMoveable()	移動の前（dispatch が呼び出される前）に呼び出される.	エージェントが移動を許可すれば true を，許可しなければ false を返す.
String getName()	リモートホストのエージェントと通信を行う時に呼び出される.	エージェントの名前を返す.

表 5.3: 本研究で追加したコールバックメソッド

5.3 電子商取引エージェントの作成

ここでは，電子商取引エージェントの雛型クラスと巡回パターンの実装クラスについて説明する．アプリケーションフレームワークでは，巡回エージェントの雛型として CustomerAgent クラスを，秘密情報管理エージェントの雛型として SecretDataManageAgent クラスを提供する．これら 2 つのクラスは abstract クラスとして定義されているため，実際に電子商取引エージェントを作成するためには，各々のクラスを継承したエージェントクラスを定義する必要がある．巡回パターンの雛型は ItineraryPattern クラスである．これを継承した ItineraryPattern1 クラス，ItineraryPattern2 クラス，ItineraryPattern3 クラスは，巡回後決済エージェント，巡回中決済エージェント，巡回交渉エージェントの各巡回パターンを実装している．以下に，各々のクラスについて説明する．プログラムコードは主要な機能だけを抜き出し，簡略化した擬似コードとして示す．

- **SecretDataManageAgent**
  秘密情報管理エージェントの雛型クラスである．継承している DispatchAgent クラス，実装している IRemoteMessageHandler インタフェースは，リモートホスト上のエージェント間通信に対応するためのものである．秘密情報管理エージェントはユーザのホストに留まり他のホストへ移動しないため，isMoveable コールバックメソッドで false を返す．その他，秘密情報を管理するために getData，setData，callMethod などのメソッドを持つ．

```
public abstract class SecretDataManageAgent extends DispatchAgent
                              implements IRemoteMessageHandler {
  protected transient RemoteMessageManager rmm;  // リモート通信
  protected SecretDataManager data;  // 秘密情報のデータベース
  protected final SecurityChecker security; //セキュリティマネージャ

  public synchronized void create() {
    // セキュリティマネージャの登録
    rmm.registerSecurity(this, security);
  }
  public boolean isMoveable() {
    return false;  // 他のホストには移動しない
  }
  public Object getData(String key) {
    return data.get(key);  // 秘密情報を取り出す
  }
  public Boolean setData(String key, Object value) {
    data.put(key, value);  // 秘密情報を格納する
  }
  public Object callMethod(String name, List args) {
    // Java のリフレクション機能を使ってメソッドを呼び出す
  }
  public void registerAgentID(byte[] id) {
```

```
    // 本エージェントにアクセス可能な巡回エージェントを登録する
  }
}
```

- **CustomerAgent**

  巡回エージェントの雛型クラスである．継承している DispatchAgent は，リモートホスト上のエージェント間通信に対応している他，ネットワーク上の自動巡回機能も持つ．ユーザのホストから移動する場合は，秘密情報管理エージェントにエージェント ID を登録する．巡回パターンクラスの arrive メソッドを呼び出すことで，巡回パターンに依存した処理を実行する．

```
public abstract class CustomerAgent extends DispatchAgent {
  // データストアや店舗の価格リストなどを持つオブジェクト
  protected CustomerController controller;
  protected ItineraryPattern itinerary;  // 巡回パターン

  public void dispatch(URL url) {
    if (isOwnerHost()) {
      // SecretDataManageAgent にエージェント ID を登録する
    }
  }
  public void arrive() {
    // 巡回パターンクラスの arrive メソッドを呼び出す
    itinerary.arrive();
  }
}
```

- **ItineraryPattern**

  巡回パターンの雛型クラスである．abstract クラスであり，各々の巡回パターンを実装するクラスは本クラスを継承しなければならない．サブクラスに実装を義務付ける arrive メソッドの他，サブクラスで共通に使われる汎用メソッドを持つ．

```
public abstract class ItineraryPattern implements Serializable {
  // データストアや店舗の価格リストなどを持つオブジェクト
  protected CustomerController controller;

  // サブクラスに実装を義務付ける arrive メソッド
  // 移動先のホストに到着した時に巡回エージェントから呼び出される
  abstract public void arrive();

  protected List getProductInfoList(AgentIdentifier shopID,
                                    String key) {
    // 仮想店舗から商品の価格リストを得る
  }
  protected boolean buyProduct(AgentIdentifier shopID,
                               List buyList) {
    // 仮想店舗・秘密情報管理エージェントと通信し，電子決済を行う
  }
```

```
  public Object callOtherAgent(AgentIdentifier aid,
                               String methodName, List args) {
    // 他のエージェントのメソッドを呼び出す
  }
}
```

- **ItineraryPattern1**

  巡回後決済エージェントを実装するクラスである．内部状態として巡回モード・決済モードを持ち，移動先のホストに到着した後，モードによって処理を切り分ける．サブクラスでは，communicate メソッドをオーバーライドするすることで，仮想店舗からの価格リスト獲得処理を変更できる．

```
public class ItineraryPattern1 extends ItineraryPattern {
  public void arrive() {  // 移動先のホストに到着
    int moveMode = getMoveMode();
    if (moveMode == MODE_ITINERANT) {
      arrive_round();     // 情報収集時の処理
    }
    else if (moveMode == MODE_PURCHASE) {
      arrive_purchase();  // 決済時の処理
    }
    agent.nextHost();     // 次のホストへ移動
  }
  protected void arrive_round() {
    communicate(...);
  }
  protected void arrive_purchase() {
    super.buyProduct(...);          // 仮想店舗と電子決済する
  }
  protected void communicate(AgentIdentifier shopID) {
    super.getProductInfoList(...);  // 価格リストを得る
  }
}
```

- **ItineraryPattern2**

  巡回中決済エージェントを実装するクラスである．ユーザの希望価格と仮想店舗の商品価格を比較し，決済を行うかどうか決定する．サブクラスでは，comparePrice メソッドをオーバーライドすることで，価格比較処理を変更できる．

```
public class ItineraryPattern2 extends ItineraryPattern {
  public void arrive() {      // 移動先のホストに到着
    communicate(...);
  }
  protected void communicate(AgentIdentifier shopID) {
    super.getProductInfoList(...);  // 価格リストを得る
    if (comparePrice(...)) {  // 価格の比較
      super.buyProduct(...);  // 仮想店舗と電子決済する
      agent.ownerHost();      // ユーザのホストへ帰る
    }
```

```
    else {
      agent.nextHost();        // 次の仮想店舗へ移動する
    }
  }
  protected boolean comparePrice(String product, String shopPrice) {
    // 価格の比較を行う
  }
}
```

- **ItineraryPattern3**

  巡回交渉エージェントを実装するクラスである．内部状態として巡回モード・決済モード・交渉モードを持ち，移動先のホストに到着した後，モードによって処理を切り分ける．サブクラスでは，negotiate メソッドをオーバーライドすることで，仮想店舗との交渉手順を変更できる．

```
public class ItineraryPattern3 extends ItineraryPattern {
  public void arrive() {  // 移動先のホストに到着
    int moveMode = getMoveMode();
    if (moveMode == MODE_ITINERANT) {
      arrive_round();     // 情報収集時の処理
    }
    else if (moveMode == MODE_PURCHASE) {
      arrive_purchase();  // 決済時の処理
    }
    else if (moveMode == MODE_NEGOTIATE) {
      arrive_negotiate();  // 交渉時の処理
    }
    agent.nextHost();      // 次のホストへ移動
  }
  protected void arrive_round() {
    communicate(...);
  }
  protected void arrive_purchase() {
    super.buyProduct(...);          // 仮想店舗と電子決済する
  }
  protected void arrive_negotiate() {
    negotiate(...);
  }
  protected void communicate(AgentIdentifier shopID) {
    super.getProductInfoList(...);  // 価格リストを得る
  }
  protected void negotiate(AgentIdentifier shopID) {
    // 仮想店舗と交渉する
  }
}
```

第 6 章

実験・考察

本章では，まず，アプリケーションフレームワークで提供される３つの巡回パターン毎に不正なホストの盗み見攻撃を想定する．次に，それに対処する電子商取引エージェントを例題として作成する．最後に，作成した例題プログラムに対して，セキュリティ機能の組み込み易さ，セキュリティ機構の問題点を考察する．プログラムコードは主要な機能だけを抜き出し，簡略化した擬似コードとして示す．

6.1 巡回後決済エージェントの例

ここでは，巡回後決済エージェントに対する不正なホストの盗み見攻撃を想定し，攻撃に対処する方法と追加するプログラムコードについて説明する．

- **想定する攻撃**
  不正な仮想店舗により，巡回エージェントが収集して来た他店舗の価格リストが盗み見される．不正な店舗は，自店の商品価格が価格リスト内の最低価格よりも低い場合に，価格を上積みして提示する．

- **保護する情報**
  各店舗から獲得する商品の価格情報

- **情報の公開先**
  - 価格情報を提供してくれる店舗のホスト
  - ユーザのホスト

- **情報の保護方法１**

- 保護手法
  巡回エージェントは，仮想店舗から獲得した商品の価格情報を，次のホストに移動する前に SSL プロトコルを用いて秘密情報管理エージェントに転送する．巡回エージェントは価格情報を移動前にクリアし，移動対象としない．
- セキュリティ
  巡回エージェントが獲得した価格情報を持たずにネットワークを巡回するため，盗み見に対処できる．秘密情報管理エージェントへの情報送信は SSL プロトコルによって保護される．

- **情報の保護方法２**
  - 保護手法
    巡回エージェントは，各店舗で獲得した商品の価格情報を，次のホストに移動する前にユーザの公開鍵で暗号化する．巡回エージェントは，暗号化した価格情報を持ってネットワーク上を巡回し，ユーザのホストに戻ってからユーザの秘密鍵で復号化する．
  - セキュリティ
    価格情報は，ユーザの公開鍵によって暗号化される．ユーザの秘密鍵を持つのはユーザのホストだけであり，暗号化された価格情報を復号化できるのはユーザだけである．

- **アプリケーションの擬似コード**
  ここでは，巡回後決済エージェントにおいて，保護手法１を用いて価格情報を保護する場合の擬似コードを示す．擬似コードでは，巡回エージェントとして Customer1 クラスを作成する．Customer1 クラスは，巡回パターンとして ExpandItineraryPattern1 クラスを使用する．ExpandItineraryPattern1 クラスは，店舗から獲得した価格リストを SSL で秘密情報管理エージェントへ登録し，価格リストをクリアした後で別のホストに移動する．巡回を終えてユーザのホストに戻ると，秘密情報管理エージェントから価格リストを取り出す．
  - **Customer1 クラスの擬似コード**

```
public class Customer1 extends CustomerAgent {
  protected void createItineraryPatternTable() {
    // ExpandItineraryPattern1 クラスを巡回パターンとして登録する
  }
}
```

  - **ExpandItineraryPattern1 クラスの擬似コード**

```
public class ExpandItineraryPattern1 extends ItineraryPattern1 {
  // communicate メソッドをオーバーライドする
  protected void communicate(AgentIdentifier shopID) {
    // 仮想店舗から価格リストを獲得
    super.communicate(shopID);
    // 秘密情報管理エージェントに店舗の価格リストを保存する
    String key = DataName.PRODUCT_LIST + "_" + shopName;
    List shops = controller.getShopManager().getAllShop();
    IDataStore store = controller.getDataStore();
    store.put(key, shops, true);
    // 店舗の価格リストをクリアする
    controller.getShopManager().removeAllShop();
  }
  // arrive_round メソッドをオーバーライドする
  protected void arrive_round() {
    if (agent.isOwnerHost()) {  // ユーザのホストに着いた場合
      // 秘密情報管理エージェントから価格リストを取り出す
      IDataStore store = controller.getDataStore();
      for (Iterator i=store.allKeys().iterator(); i.hasNext(); ) {
        String key = (String)i.next();
        // 価格リストを検索する
        if (key.startsWith(DataName.PRODUCT_LIST)) {
          DataInfo dInfo = (DataInfo)store.get(key);
          List shops = (List)dInfo.getData();
          // 価格リストとして登録する
          controller.getShopManager().addShop(shops);
        }
      }
    }
    super.arrive_round();
  }
}
```

- **セキュリティポリシーの定義**
  ここでは，セキュリティポリシーの追加分だけを示す．巡回エージェントから登録・参照される各店舗の価格リストへのアクセスポリシーおよび秘密情報管理エージェントの getAllKey メソッドを呼び出すためのポリシーを定義している．

```
<data>  <!-- shop1 の価格リスト -->
  <data-name>PRODUCT_LIST_shop1</data-name>
  <protect name="ssl" />
  <!-- ユーザのホスト上で価格リストを参照する -->
  <host name="192.168.1.1">
    <agent name="CN=customer, OU=..... ">
      <phase name="owner">
        <access name="dataget" />
      </phase>
    </agent>
  </host>
  <!-- shop1 のホスト上から価格リストを登録する -->
  <host name="192.168.1.2">
    <agent name="CN=shop1, OU=..... ">
      <phase name="itinerant">
        <access name="dataset" />
      </phase>
    </agent>
  </host>
```

```
</data>
<data>  <!-- shop2 の価格リスト -->
  <data-name>PRODUCT_LIST_shop2</data-name>
  .....
</data>
<method>  <!-- メソッド呼び出し -->
  <method-name>getAllKey</method-name>
  <protect name="unguarded" />
  <!-- ユーザのホスト上でメソッドを呼び出す -->
  <host name="192.168.1.1">
    <agent name="CN=customer, OU=..... ">
      <phase name="owner">
        <access name="call" />
      </phase>
    </agent>
  </host>
</method>
```

6.2 巡回中決済エージェントの例

ここでは，巡回中決済エージェントに対する不正なホストの盗み見攻撃を想定し，攻撃に対処する方法と追加するプログラムコードについて説明する．

- **想定する攻撃**
  不正なホストにより，巡回エージェントが保持しているユーザの希望価格が盗み見される．不正なホストは，ユーザの希望価格の範囲内で，本来の提示価格よりも高い価格を提示する．

- **保護する情報**
  ユーザの希望価格

- **情報の公開先**
  ユーザのホスト

- **情報の保護方法**

  – 保護手法
    巡回エージェントは，店舗の提示価格とユーザの希望価格をユーザのホスト上で比較する．このために秘密情報管理エージェントに価格比較機能を追加する．巡回エージェントはユーザの希望価格を持たず，店舗の提示価格を引数として秘密情報管理エージェントから価格比較の結果だけを受け取る．

– セキュリティ
巡回エージェントがユーザの希望価格を持たずにネットワークを巡回するため，盗み見に対処できる．

- **アプリケーションの擬似コード**

ここでは，巡回中決済エージェントにおいて，ユーザの希望価格を保護する場合の擬似コードを示す．擬似コードでは，巡回エージェントとして Customer2 クラスを作成する．Customer2 クラスは，巡回パターンとして ExpandItineraryPattern2 クラスを使用する．ExpandItineraryPattern2 は，店舗の提示価格を引数として秘密情報管理エージェントの価格比較メソッドを呼び出す．秘密情報管理エージェントとして SecretData2 クラスを作成し，価格比較関数を実装する．

– **Customer2 クラスの擬似コード**

```
public class Customer2 extends CustomerAgent {
  public void dispatch(URL url) {
    super.dispatch(url);
    if (super.isOwnerHost()) {{  // ユーザのホストから離れる場合
      // ユーザの希望商品と希望価格を取得する
      String product = super.productField.getText();
      String price = super.priceField.getText();
      // 希望商品と希望価格を秘密情報管理エージェントに格納する
      IDataStore store = controller.getDataStore();
      store.put(DataName.HOPE_PRODUCT, product, true);
      store.put(DataName.HOPE_PRICE, price, true);
      // 希望価格をクリアする
      super.priceField.setText("");
    }
  }
  protected void createItineraryPatternTable() {
    // ExpandItineraryPattern2 クラスを巡回パターンとして登録する
  }
}
```

– **ExpandItineraryPattern2 クラスの擬似コード**

```
public class ExpandItineraryPattern2 extends ItineraryPattern2 {
  // comparePrice メソッドをオーバーライドして，
  // 秘密情報管理エージェントの comparePrice メソッドを呼び出す
  protected boolean comparePrice(String product, String shopPrice) {
    // メソッド呼び出し用の情報を作成
    ProductData pData = new ProductData(product, shopPrice, "1");
    List args = new ArrayList();
    args.add(pData);
    CallMethod callInfo = new CallMethod("comparePrice", args);
    // メソッドの呼び出し
    IDataStore store = controller.getDataStore();
    MessagePacket retPacket = store.method(MessagePacket.SSL,
                                           callInfo);
    // 価格比較結果の取り出し
    Boolean result = (Boolean)retPacket.getMessage();
    return result.booleanValue();
```

```
  }
}
```

– **SecretData2 クラスの擬似コード**

```
public class SecretData2 extends SecretDataManageAgent {
  // 価格比較を行う comparePrice メソッドを追加する
  public Boolean comparePrice(ProductData pData) {
    // 店舗の提示商品名・価格を取り出す
    String product = pData.getName();
    String price = pData.getPrice();
    // ユーザの希望商品名・希望価格を取り出す
    String hopeProduct = (String)super.data.get(
                                        DataName.HOPE_PRODUCT);
    String hopePrice = (String)super.data.get(DataName.HOPE_PRICE);
    // 商品名と価格を比較する
    boolean result = false;
    if (hopeProduct.equals(product)) {
      boolean comp = StringUtility.numberStringGT(price, hopePrice);
      result = !comp;
    }
    return new Boolean(result);
  }
}
```

- **セキュリティポリシーの定義**

  ここでは，セキュリティポリシーの追加分だけを示す．ユーザの希望商品・希望価格を秘密情報管理エージェントに登録するためのポリシーおよび秘密情報管理エージェントの comparePrice メソッドを呼び出すためのポリシーを定義している．

```
<data>  <!-- 希望商品・希望価格の登録 -->
  <data-name>HOPE_PRODUCT</data-name>
  <data-name>HOPE_PRICE</data-name>
  <protect name="unguarded" />
  <!-- ユーザのホスト上で情報を登録する -->
  <host name="192.168.1.1">
    <agent name="CN=customer, OU=..... ">
      <phase name="itinerant">
        <access name="dataset" />
      </phase>
    </agent>
  </host>
 </data>
<method>  <!-- メソッド呼び出し -->
  <method-name>comparePrice</method-name>
  <protect name="ssl" />
  <!-- shop1 のホスト上からメソッドを呼び出す -->
  <host name="192.168.1.2">
    <agent name="CN=shop1, OU=..... ">
      <phase name="itinerant">
        <access name="call" />
      </phase>
    </agent>
  </host>
  <!-- shop2 のホスト上からメソッドを呼び出す -->
```

```
  <host name="192.168.1.3">
    <agent name="CN=shop2, OU=..... ">
      <phase name="itinerant">
        <access name="call" />
      </phase>
    </agent>
  </host>
</method>
```

6.3 巡回交渉エージェントの例

ここでは，巡回交渉エージェントに対する不正なホストの盗み見攻撃を想定し，攻撃に対処する方法と追加するプログラムコードについて説明する．

- **想定する攻撃**
  不正なホストが巡回エージェントの交渉手順を盗み見，自分に有利となるように交渉を進める．
- **保護する情報**
  巡回エージェントの交渉手順
- **情報の公開先**
  ユーザのホスト
- **情報の保護方法**
  - 保護手法
    巡回エージェントは交渉対象のホストに移動して仮想店舗と交渉を行うが，交渉手順は秘密情報管理エージェントが管理する．巡回エージェントは，仮想店舗との間で行うべき交渉の手順を秘密情報管理エージェントに問い合わせながら交渉を進める．
  - セキュリティ
    巡回エージェントが交渉手順を持たずに交渉先のホストに移動するため，盗み見に対処できる．交渉対象の仮想店舗は，交渉が完了した時点で手順を把握できるが，事前に手順を知ることはできない．
- **交渉手順**
  今回実装した交渉手順は，仮想店舗との間で価格交渉を行うものである．巡回エージェントは複数の店舗の価格リストを収集してユーザのホストに戻り，価格の低い

方から店舗を２つ選ぶ．その後，２番目に価格の低かった店舗に移動し，最も価格の低かった店舗の情報を用いて価格交渉（値下げ交渉）を行う．交渉が成立すればその店舗で決済を行い，失敗すれば，最も価格の低かった店舗に移動して決済を行う．

- **アプリケーションの擬似コード**

  ここでは，巡回交渉エージェントにおいて，仮想店舗との交渉手順を保護する場合の擬似コードを示す．擬似コードでは，巡回エージェントとして Customer3 クラスを作成する．Customer3 クラスは，巡回パターンとして ExpandItineraryPattern3 クラスを使用する．ExpandItineraryPattern3 クラスは，仮想店舗と交渉を行う際に，実行すべき処理手順を秘密情報管理エージェントに問い合わせる．秘密情報管理エージェントとして SecretData3 クラスを作成し，Customer3 クラスからの交渉手順の問い合わせに対応する．

  - **Customer3 クラスの擬似コード**

```
public class Customer3 extends CustomerAgent {
  public void dispatch(URL url) {
    // 仮想店舗へ交渉に向う場合
    if (isOwnerHost() && (controller.getItinerantMode() ==
                          ItineraryPattern3.MODE_NEGOTIATE)) {
      // 交渉情報を秘密譲歩管理エージェントに登録する
      NegotiateInfo info = controller.getNegotiateInfo();
      IDataStore store = controller.getDataStore();
      store.put(DataName.NEGOTIATE_INFO, info, true);
      // 自分が持っている交渉情報をクリアする
      controller.setNegotiateInfo(null);
    }
    super.dispatch(url);  // デフォルトの処理
  }
  protected void createItineraryPatternTable() {
    // ExpandItineraryPattern2 クラスを巡回パターンとして登録する
  }
}
```

  - **ExpandItineraryPattern3 クラスの擬似コード**

```
public class ExpandItineraryPattern3 extends ItineraryPattern3 {
  // negotiate メソッドをオーバーライドする
  protected void negotiate(AgentIdentifier shopID) {
    NegotiateOperation op = new NegotiateOperation();
    while (true) {
      // 秘密情報管理エージェントの negotiate メソッドを呼び出して
      // 交渉手順を獲得する
      op = getNegotiateOperation(op);
      // 交渉手順を実行する
      Object ret = doOperation(op);
      // 交渉手順が無くなれば交渉を終了する
      if (!op.isNeedNext() || (ret == null)) {
        break;
      }
      op.setReturnValue(ret);
```

```
    }
  }
  private NegotiateOperation getNegotiateOperation(
                                  NegotiateOperation oldOP) {
    // メソッド呼び出し用の情報を作成
    URL url = agent.getAgentContext().getCurrentHost();
    String currentIP = StringUtility.UrlToIP(url.toString());
    List args = new ArrayList();
    args.add(currentIP);
    args.add(oldOP);
    CallMethod callInfo = new CallMethod("negotiate", args);
    // メソッドを呼び出す
    IDataStore store = controller.getDataStore();
    MessagePacket retPacket = store.method(MessagePacket.SSL,
                                           callInfo);
    // 交渉手順を取り出す
    Object retInfo = retPacket.getMessage();
    return retInfo;
  }
  private Object doOperation(NegotiateOperation op) {
    Object retVal = null;
    String method = op.getMethodName();
    List args = op.getMethodParam();
    if (op.isShopMethod()) {  // 仮想店舗のメソッドを呼び出す
      retVal = super.callOtherAgent(shopID, method, args);
    }
    else {  // このクラス内のメソッドを呼び出す
      retVal = ReflectUtility.invokeMethod(this, method, args);
    }
    return retVal;
  }
}
```

– **SecretData3 クラスの擬似コード**

```
public class SecretData3 extends SecretDataManageAgent {
  public NegotiateOperation negotiate(String ip,
                                      NegotiateOperation oldOP) {
    NegotiateOperation op = null;
    switch (step) {  // 次の交渉手順を返す
    case STEP0:
         op = step0(ip, oldOP); break;
    case STEP1:
         op = step1(ip, oldOP); break;
    default: break;
    }
    return op;
  }
  private NegotiateOperation step0(String ip,
                                   NegotiateOperation oldOP) {
    // 巡回エージェントに仮想店舗の negotiate メソッドを
    // 呼ばせるための手順を返す
  }
  private NegotiateOperation step1(String ip,
                                   NegotiateOperation oldOP) {
    // 巡回エージェントに決済またはユーザのホストへの移動を
    // させるための手順を返す
  }
```

```
}
```

- **セキュリティポリシーの定義**
  ここでは，セキュリティポリシーの追加分だけを示す．

```
<data> <!-- 交渉情報の登録 -->
  <data-name>NEGOTIATE_INFO</data-name>
  <protect name="unduarded" />
  <!-- ユーザのホスト上で情報を登録する -->
  <host name="192.168.1.1">
    <agent name="CN=customer, OU=..... ">
      <phase name="owner">
        <access name="dataset" />
      </phase>
    </agent>
  </host>
</data>
<method> <!-- メソッド呼び出し -->
  <method-name>negotiate</method-name>
  <protect name="ssl" />
  <!-- shop1 のホスト上でメソッドを呼び出す -->
  <host name="192.168.1.2">
    <agent name="CN=shop1, OU=..... ">
      <phase name="negotiate">
        <access name="call" />
      </phase>
    </agent>
  </host>
  <!-- shop2 のホスト上でメソッドを呼び出す -->
  <host name="192.168.1.3">
    <agent name="CN=shop2, OU=..... ">
      <phase name="negotiate">
        <access name="call" />
      </phase>
    </agent>
  </host>
</method>
```

6.4 考察

6.4.1 セキュリティ機能の組み込み易さ

例題として作成した３つの電子商取引エージェントを，本アプリケーションフレームワークを用いずに AgentSpace 上で作成する場合，各々のエージェントは表 6.1に示す機能を実装する必要がある．表中のステップ数は，フレームワークで各々の機能を実装するためにかかったプログラムコードの行数である．SSL プロトコル，RSA による暗号化，SHA-1 によるメッセージダイジェストの生成などの処理は，JCE や JSSE により実現できるが，フレームワークのライブラリを使用するよりも煩雑になる．Aglets や Voyager な

どのエージェントシステムでは，ネットワーク上の自動巡回機能やリモートホスト上のエージェント間通信機能自体はサポートされる．

分類	機能	ステップ数
巡回エージェントの機能	ネットワーク上の自動巡回機能	300
	ネットワーク上の巡回パターン	650
	データストアの機能	350
秘密情報管理エージェントの機能	秘密情報を管理する機能	350
	セキュリティマネージャの機能	850
	外部ファイルに定義されているセキュリティポリシーを読み込んで適用する機能	800
その他	リモートホスト上のエージェント間でセキュアな通信を行う機能	800
	SET プロトコルを用いてクレジットカードの情報を保護するための機能	1,100
	暗号アルゴリズム・暗号プロトコル・電子署名などの基本的なセキュリティ機能	600
	合計	5,800

表 6.1: フレームワークを用いない場合に実装する機能

ここでは，アプリケーションフレームワークが提供する機能だけを挙げたが，この他にも当然，交渉手順などのアプリケーション固有のロジックやセキュリティポリシーを記述する必要がある．フレームワークのプログラムコードは汎用性を考慮したものであるため，アプリケーションに特化した形でプログラムを作成すれば，表 6.1のステップ数よりも若干コード量は少なくなると考えられる．しかし例題ではフレームワークを用いることで，盗み見に対処した巡回パターンの異なる電子商取引エージェントを少ない記述量で容易に作成することができた．以下に本アプリケーションフレームワークを用いて電子商取引エージェントを作成する場合の利点を述べる．

- **エージェント本体と秘密情報の分離**
  巡回エージェントおよび秘密情報管理エージェントの雛型クラスを継承することで，エージェント本体と秘密情報を分離するための機構が得られる．

- **セキュリティポリシーの定義**
  アプリケーション毎にセキュリティポリシーを定義できる．ポリシーは外部のポリシーファイルに定義するため，電子商取引エージェントのアプリケーションロジックと混在しない．

- **セキュリティ機能の強制的な組み込み**
  巡回エージェントから秘密情報管理エージェントへのアクセスは，ポリシーファイルに定義されているものだけが許可される．これにより，セキュリティポリシーを定義し忘れた場合やプログラムのバグにより不正なアクセスを行ってしまった場合でも，秘密情報を保護できる．

- **リモートホスト間のエージェント通信**
  巡回エージェントは，フレームワークのデータストア機能を使用して秘密情報管理エージェントへ要求を送る．これにより，巡回エージェントでは，ネットワーク通信を意識することなく秘密情報を保護することができる．次のプログラムコードは，巡回エージェントが秘密情報管理エージェントへ要求を送信するためのものである．コード中の store はデータストアのオブジェクトである．

  – データを秘密情報管理エージェントから取り出す

  ```
  Object data = store.get(dataName);
  ```

  – データを秘密情報管理エージェントに登録する

  ```
  store.put(dataName, data, true);
  ```

  – 秘密情報管理エージェントのメソッドを呼び出す

  ```
  CallMethod callInfo = new CallMethod("methodName", args);
  String protocol = "ssl";
  Object data = store.method(protocol, callInfo);
  ```

- **SET プロトコルの使用**
  クレジットカードの情報を SET プロトコルにより保護できる．

- **巡回パターン**
  巡回後決済エージェント・巡回中決済エージェント・巡回交渉エージェントの雛型クラスを継承することで，各々の巡回パターンを実装できる．

- **アプリケーションロジックの追加**
  各々の巡回パターンクラスでは，コールバックメソッドをオーバーライドすることで，容易にアプリケーションロジックを追加できる．

6.4.2 セキュリティ機構の問題点

提案したセキュリティ機構の問題点を以下に挙げる．

- ユーザのホストの回線切断が制限される
- セキュリティ機能を追加することで実行時間のオーバーヘッドが増える
- 不正なホストによる巡回エージェントへのなりすまし

まず，ユーザのホストの回線切断についてであるが，フレームワークを用いて作成される電子商取引エージェントでは，巡回エージェントと秘密情報管理エージェントの間で通信が行われている間，ユーザのホストと仮想店舗のあるホストは共にネットワークに接続されていなければならない．ただし，巡回エージェントはユーザのホストおよび仮想店舗のホストと非同期に実行されるため，それぞれのホストはネットワークに常に接続されていなければならない．仮想店舗のあるホストはサービスを提供するサーバであり，ネットワークに常時接続されている場合が多いと想定される．これに対しユーザは，PDA などの携帯端末から電子商取引エージェントを用いることも考えられ，一般的に常時接続の環境を仮定できない．本研究では，ユーザのホストを信頼できるホストと仮定し，秘密情報管理エージェントはユーザのホストから移動しないものとしたが，ネットワークへの常時接続の環境を持つ信頼できるサーバが他に仮定できる場合は，秘密情報管理エージェントをそのサーバへ移動させ，巡回エージェントからの要求を監視するような運用形態も考えられる．このような運用形態では，ユーザのホストの回線切断が可能になる．秘密情報管理エージェントはモバイルエージェントであるため容易に信頼できるサーバへ移動できるが，移動したことを巡回エージェントに知らせるための仕組みが必要になる．

実行時間のオーバーヘッドは，ユーザのホストおよび仮想店舗のホストのマシンパワーに大きく左右される．これは，巡回エージェントと秘密情報管理エージェント間の通信を，SSL や SET などの暗号プロトコルを用いて保護しているためである．各々の電子商取引エージェントで本当に守らなければならないものだけを秘密情報とし，エージェント間の通信回数を減らすことで，巡回エージェントがユーザから与えられた仕事を終えるまでの時間を減らすことができる．フレームワークでは，アプリケーション毎に異なるセキュリティポリシーを設定できるようにすることでこれを可能にしている．

なりすましの問題は，移動先の不正なホストが巡回エージェントのプログラムを解析し，巡回エージェントになりすまして秘密情報管エージェントにアクセスすることである．一般的に，なりすましの問題に対しては認証が用いられるが，本セキュリティ機構では，巡回エージェントの認証情報が移動先のホストによって盗み見されてしまう危険性がある．このため，Time Limited Blackbox Security 手法を用いて巡回エージェントを難読化する必要がある．ただし，現在，難読化には絶対的なアルゴリズムが存在しないため，難読化されたエージェントを解析するのに必要と仮定される時間をエージェントのライフタイムに設定し，ライフタイムの間だけエージェントを有効にしなければならない．更に巡回エージェントにランダムで一意な ID を割り当て，ライフタイムと組み合わせて秘密情報管理エージェントへのアクセスを制限しなければならない．フレームワークには，巡回エージェントにライフタイムを設定する機能，ランダムな ID を割り振りライフタイムと合わせてアクセス制限を行う機能が共に用意されている．このため，ライフタイムを明確に定義できると仮定すれば，巡回エージェントの難読化を行うことでなりすましに対処できる．

第 7 章

まとめ

本章では，本システムの特徴と今後の課題について述べる．

7.1 本システムの特徴

本研究で提案したセキュリティ機構は，モバイルエージェントの秘密情報を不正なホストの盗み見から守るものである．巡回エージェントと秘密情報管理エージェントを用いて秘密情報をエージェント本体から分離し，秘密情報へのアクセスを制限することでセキュリティを確保している．この手法では，巡回エージェントは秘密情報が必要になる度に秘密情報管理エージェントに要求を送る必要があるため，プログラムコードが煩雑になり易い．このため，アプリケーションとして電子商取引エージェントを考え，セキュリティ機構をアプリケーションフレームワークとし実現した．本アプリケーションフレームワークを用いることにより，巡回エージェントは秘密情報管理エージェントとの間でセキュアな通信を容易に実現できる．また，セキュリティポリシーをカスタマイズすることで，アプリケーション毎に異なるセキュリティ要件を適用でき，柔軟なセキュリティ対策が可能である．セキュリティポリシーは外部ファイルに XML を用いて定義するため，アプリケーションロジックとは分離される．

7.2 今後の課題

今後の課題としては以下の様なことが挙げられる．

- 不正なホストによる改竄への対処
  電子商取引エージェントを実用的なシステムで使用するためには，不正なホストに

よる改竄の問題も考慮しなければならない．ソフトウェアだけで無差別な改竄を防ぐのは非常に困難であるが，エージェントプログラムを解析して何らかの目的を持って行われる改竄には，Time Limited Blackbox Security 手法や実行トレースを用いた手法などと組み合わせ，対処していく必要がある．

- 他のアプリケーションへの応用
  本研究では，具体的なアプリケーションとして，複数のホストを巡回する電子商取引エージェントを考えたが，モバイルエージェントは他のアプリケーションでも使用が期待されている．提案したセキュリティ機構はシンプルな構造であるため，オークションや電子商取引以外のアプリケーションへも応用できる．

謝辞

指導教官の渡部卓雄先生には，自由な研究環境をはじめ本研究の全過程を通して親切な指導と助言をして頂きました．深く感謝の意を表し，心より御礼申し上げます．株式会社富士通北陸システムズには，企業派遣の学生として研究の機会を与えて頂きました．また留学中も学業や研究に専念できるよう常に配慮して頂き，心から感謝致します．天野憲樹先生には，本研究についての適切な技術的御指導と御助言を頂きました．厚く御礼申し上げます．二木厚吉先生をはじめとする言語設計学講座の皆様には，セミナーの際に色々と貴重な意見を頂きました．深く感謝致します．佐藤一郎助教授には，快く AgentSpace を利用することを許可して頂きました．感謝の意を表します．

参考文献

[1] David M. Chess.:Security Issue in Mobile Code Systems, Lecture Notes in Computer Science Vol.1419, pp1-14, Springer(1998).

[2] 本位田真一, 飯島正, 大須賀昭彦.:エージェント技術, 共立出版, 1999.

[3] Fritz Hohl.:Time Limited Blackbox Security:Protecting Mobile Agents from Malicious Hosts, Lecture Notes in Computer Science Vol.1419, pp92-113, Springer(1998).

[4] Fritz Hohl.:A Protocol to Detect Malicious Hosts Attacks by Using Reference States, Technical Report Nr, 1999.

[5] 岩井俊弥.:Java モバイル・エージェント, ソフト・リサーチセンター, 1998.

[6] Dejan Milojicic et al.:MASIF The OMG Mobile Agent System Interoperability Facility, Lecture Notes in Computer Science Vol.1477, pp92-113, Springer(1998).

[7] 山崎重一郎, 津田宏.:Telescript 言語入門, アスキー出版局, 1996.

[8] 佐藤一郎.:モバイルエージェントの動向, 人工知能学会誌, Vol.14, No.4, pp598-605(1999).

[9] http://www.trl.ibm.co.jp/aglets/

[10] 小野康一.:移動エージェント Aglets のセキュリティ・モデル, コンピュータソフトウェア, Vol.17, No.4, pp2-14(2000).

[11] http://www.objectspace.com/products/voyager/

[12] http://www2.toshiba.co.jp/plangent/

[13] 田原康之, 長谷川哲夫, 大須賀昭彦, 本位田真一.:知的ネットワークエージェント Plangent のセキュリティ・セーフティ, インターネットテクノロジーワークショップ, 1998.

[14] http://www.islab.is.ocha.ac.jp/agent/

[15] 佐藤一郎, 高橋美奈子, 棚橋杏子, 吉野裕子.:モ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, 1998.

[16] Tomas Sander,Christain F. Tschudin.:Protecting Mobile Agents Against Malicious Hosts, Lecture Notes in Computer Science Vol.1419, pp44-60, Springer(1998).

[17] 岩井俊弥, 栗原健,Wu Wen, 溝口文雄.:耐タンパ・移動エージェントの調査研究, 第 18 回 IPA 技術発表会, 1999.

[18] Giovanni Vigna.:Cryptographic Trace for Mobile Agents, Lecture Notes in Computer Science Vol.1419, pp137-153, Springer(1998).

第 A 章

付録

ここでは，本アプリケーションフレームワークおよび作成した例題アプリケーションを動作させるための，実行環境，インストール方法，実行方法について簡単に説明する．

A.1　実行環境

- OS：Windows98/2000
- JDK：JDK1.3
- Java 拡張機能：JCE1.2.1，JSSE1.0.2，Java Project X Technology Release2
- エージェントシステム：AgentSpace（1999 年 4 月 30 日版以前のバージョン）

A.2　インストール方法

電子商取引エージェントを実行する全てのホストでは，JDK，JCE，JSSE，Java Project X，AgentSpace を以下の URL から入手し，インストールしておく必要がある．

```
http://jdc.sun.co.jp/
http://java.sun.com/products/jce/
http://java.sun.com/products/jsse/
http://developer.java.sun.com/developer/products/xml/
http://islab.is.ocha.ac.jp/agent
```

次に，以下の手順でアプリケーションフレームワークのインストールおよび環境設定を行う．

- アプリケーションフレームワークのインストール
  以下のファイルを任意のディレクトリにダウンロードし，解凍する．
  `http://www.jaist.ac.jp/~smurata/Framework.zip`
  表 A.1に従い，生成されたディレクトリをコピーする．各々のホストでのコピー先は，AgentSpace のインストール先の system ディレクトリである．

ディレクトリ	コピー先
ECSystem-customer	ユーザのホストにコピーし，ECSystem に変名する．
ECSystem-shop	仮想店舗のホストにコピーし，ECSystem に変名する．
ECSystem-pg	Payment Gateway のホストにコピーし，ECSystem に変名する．
agentspace	ディレクトリ内の全てのファイルを，全ホストの system/agentspace ディレクトリに上書きでコピーする．
keystore	各ホストに証明書をインストールするための雛型ファイル．
ECSystem.zip	開発環境一式をまとめたアーカイブ．

表 A.1: 生成されたディレクトリのコピー

- 証明書のインストール
  生成された`keystore/CertInstall.txt` の内容に従って，ユーザ，仮想店舗，PaymentGateway 用の証明書およびキーストアを生成する．本研究では，ホストの証明書に Verisign 社のテスト用サーバ ID を使用した．

- 環境設定
  各々のホストで以下のように環境設定を行う．ユーザ名および仮想店舗名とは，証明書を生成する時に使用したユーザおよび仮想店舗の名前である．カレントディレクトリは，AgentSpace の system ディレクトリとしている．

  - ユーザのホスト
    `./ECSystem/data/policy/policy.xml` に定義されている仮想店舗のホストの IP アドレスと仮想店舗名を変更する．更に
    `./ECSystem/data/customer/customer.data` に定義されているユーザ名を変更する．
  - 仮想店舗のホスト
    `./ECSystem/data/keystore/access_info_shop.data` および

./ECSystem/data/keystore/access_info.data で，キーストアにアクセスするための情報を設定する．更に./ECSystem/data/shop/shop.data に定義されている PaymentGateway の IP アドレスを変更する．

– PaymentGateway のホスト
./ECSystem/data/policy/policy_payment-gateway.xml に定義されている仮想店舗のホストの IP アドレスと仮想店舗名を変更する．

A.3 実行方法

AgentSpace の system ディレクトリで以下のように入力し，AgentSpace を起動する．
java agentspace.AgentServer
AgentSpace が起動したら，それぞれのホストでは以下の手順でエージェントを起動する．カレントディレクトリは，AgentSpace の system ディレクトリとしている．

- ユーザのホスト
./ECSystem/agent/SecretData.agent を Load して秘密情報管理エージェントを起動した後，./ECSystem/agent/Customer.agent を Load して巡回エージェント図 A.1を起動する．

図 A.1: 巡回エージェント

- 仮想店舗のホスト

  ./ECSystem/agent/Merchant.agent を Load して仮想店舗図 A.2を起動する．エージェントが起動したら，読み込みボタンを押し，データファイルを指定する．

  ./ECSystem/data/shop にデフォルトのデータファイルが用意されている．

図 A.2: 仮想店舗

- PaymentGateway のホスト

  ./ECSystem/agent/PaymentGateway.agent を Load する．

全てのエージェントが起動したら，巡回エージェントの店入力ボタンを押し，移動先のホストを指定する．出発ボタンを押すとエージェントは巡回を開始する．巡回後決済エージェント，巡回交渉エージェントでは，エージェントが情報収集して戻った後，店情報ボタンを押して収集情報を表示させる．商品リストから目的の商品を選択し，決済または交渉を行う．

例題として作成したエージェントのファイル名を表 A.2に示す．それぞれの巡回エージェントを実行するためには，対応する秘密情報管理エージェントを先に起動しておく必要がある．

ファイル名	機能	対応
Customer.agent	フレームワークで提供される機能だけを持つ巡回エージェント．決済情報だけを保護する．	SecretData
Customer1.agent	巡回後決済エージェントの例題	SecretData
Customer2.agent	巡回中決済エージェントの例題	SecretData2
Customer3.agent	巡回交渉エージェントの例題	SecretData3
SecretData.agent	フレームワークで提供される機能だけを持つ秘密情報管理エージェント	—
SecretData2.agent	Customer2 用の秘密情報管理エージェント	—
SecretData3.agent	Customer3 用の秘密情報管理エージェント	—
PaymentGateway.agent	カード認証機関エージェント	—
Merchant.agent	仮想店舗エージェント	—

表 A.2: エージェントファイルの説明