クラスを伴って移動するオブジェクトのセキュリティの実現

Implementation of Security System of Mobile Object with Classes Delivered on Demand

吉田泰光, 塚本享治

要　約　オブジェクト指向技術を使って，インターネットのようなネットワーク環境での分散システムの開発と稼働を容易にするための共通の環境を実現したシステムがオブジェクト指向分散環境 OZ である．

OZ では，分散システム間でオブジェクトを移動することができ，更に，そのオブジェクトの稼働に必要なクラスが配送される．しかし，配送されてきたクラスは未知であり，その実行にはセキュリティ上の危険が伴う．

この問題を解決するため，移動してきたオブジェクトとそうでないオブジェクトにそれぞれ赤と緑の色を付けて区別し，その色によってアクセス制御を行う．これによって，セキュリティを確保しながら配送されてきた未知のクラスを安全に実行できる．本稿では，この仕組みと Java による実現について述べる．

Abstract　Object-Oriented Distributed Systems Environment OZ is a system by means of which facilitates to develop and operate a distributed system in a networking environment like the Internet, applying the object-oriented approach.

In System OZ, an object is transferred among distributed systems, and classes required for operating the object are delivered automatically on demand. However, delivered classes are unknown to the system, so that a violation of the security policy of the system may occur when executing its classes.

To resolve security problems, System OZ have individual objects assigned a color either red or green according to whether the object comes from outside or not, and performs access controls according to the assigned color. This mechanism ensures that the system executes unknown classes delivered while maintaining the system security.

This paper reports this security mechanism and its implementation in Java.

1. はじめに

インターネットに見られるように，単一のシステムだけでなく様々な種類のシステムが相互に接続され，それらのシステムが連係して運用されるようになってきている．このような単一の計算機を越えた範囲で利用されることを前提に作成されたシステムを分散システムと呼ぶ．また，分散システム相互の連係した運用を相互運用と呼ぶ．World Wide Web に見られる，サーバ，プロキシ，検索エンジン，ブラウザなどは一つの分散システムの例として挙げることができる．

これらの分散システムは，RFC[*1] に従って作成されてはいるが，一般には，使用されるソフトウェアは個々に開発，あるいは，管理されている．また，独立したネットワークが相互に接続されているインターネットのような環境では，従来のような同期したソフトウェアの一斉変更は現実的ではない．この結果，個々にソフトウェアの

拡張や改良を行うことになり，分散システム間で使用されるソフトウェアに差異が生じてくる．この差異のために分散システム間の連係が崩れることがある．このように相互運用性を保ったままでソフトウェアの拡張や改良を行うことは難しい．一方，インターネットの普及にともなって，これらのソフトウェアへの拡張や改良の要求は高まるばかりである．

既存の動作をそのままにソフトウェアの拡張や改良を可能にする技術としてオブジェクト指向技術がある．つまり，分散システム間の接続インタフェースを抽象化したクラスを使って記述することにより，個々の分散システムはそのクラスの継承を使って既存のクラス（ソフトウェア）の拡張や改良を行うことができる．接続インタフェースにクラスが含まれることは，分散システム間でオブジェクトそのものを送受信できることを意味する．そして，送受信したオブジェクトが稼働するために必要なクラスが，その分散システムに必ずしも存在するとは限らない．従って，送受信したオブジェクトの稼働に必要なクラスを必要に応じて配送する必要がある．オブジェクト送受信，すなわち，オブジェクトの移動とそれに伴ってクラスを配送することにより，相互運用性を確保したソフトウェアの拡張や改良が可能になる．

しかし，一般に配送されてくるクラスは未知であり，その実行にはセキュリティ上の危険が伴うことになる．配送されたソフトウェアの実行時のセキュリティ確保のモデルとして，SandBox モデルがある．SandBox モデルでは，配送されたソフトウェアを SandBox の中に隔離してアクセス制御を行う．SandBox モデルは，クライアントサーバ方式にみられるようなアプリケーションとして完結しているソフトウェアに対しては有効である．一方，他のオブジェクトや分散システムとの連係を前提とするソフトウェアでは，プログラミング上の自由度が問題となる．

また，個々の分散システムは，異なる組織によって運用されており，その運用方針は様々である．従って，エンドユーザに提供するセキュリティは個々の分散システムによって異なる．個々の分散システムがその運用方針に従ったセキュリティを構築できる仕組みが必要となる．

本稿では，クラスを伴って移動するオブジェクトからシステムを保護するための基本となるセキュリティの仕組みと Java による実現について述べる．

2. オブジェクト指向分散環境 OZ の概要

ネットワーク環境における分散システムの開発では，相互運用性を確保したままでソフトウェアの拡張や改良を行うことが難しい．ネットワーク環境にオブジェクト指向技術を適用することにより，この問題を解決しようというプロジェクト[1][2]が 1992 年から実施された．このプロジェクトの特徴は，オブジェクト自身だけでなく，そのオブジェクトが稼働するために必要なクラスをも配送する稼働環境とそのための開発環境の研究開発にある．この一連の研究開発で作成されたシステムが，オブジェクト指向分散環境 OZ[*2] である．

OZ は，ネットワーク環境にオブジェクト指向技術を全面的に適用し，統一されたモデルによる共通の分散システムの環境を提供する．その結果として，開発時に必要となるソフトウェアの共有と稼働時に必要となるソフトウェアの流通を実現し，ネッ

トワーク環境でのソフトウェアの相互利用を可能にしている．

2.1 オブジェクトモデル

一般に，オブジェクト指向では，全てのオブジェクトは相互に等しくメソッドを呼び出すことができる．しかし，OZ ではオブジェクト指向技術をネットワーク環境に適用する時，ネットワーク側から識別できるオブジェクトとネットワーク側から識別できない限定された範囲で識別できるオブジェクトの二つに分けた．

ネットワーク側から識別できるオブジェクトのクラスの作成には，並行実行，セキュリティ，一貫性，例外処理，タイムアウトなどを考慮したネットワークに特有のプログラミング技術が必要となる．一方，ネットワーク側から識別されないオブジェクトのクラスの作成には，多くの場合はそれらを考慮する必要がなく通常のプログラミング技術で十分である．前者をグローバルオブジェクト，後者をローカルオブジェクトと呼ぶ．また，一つのグローバルオブジェクトと複数のローカルオブジェクトの集まりをセルと呼ぶ．このオブジェクトモデルを図 1 に示す．

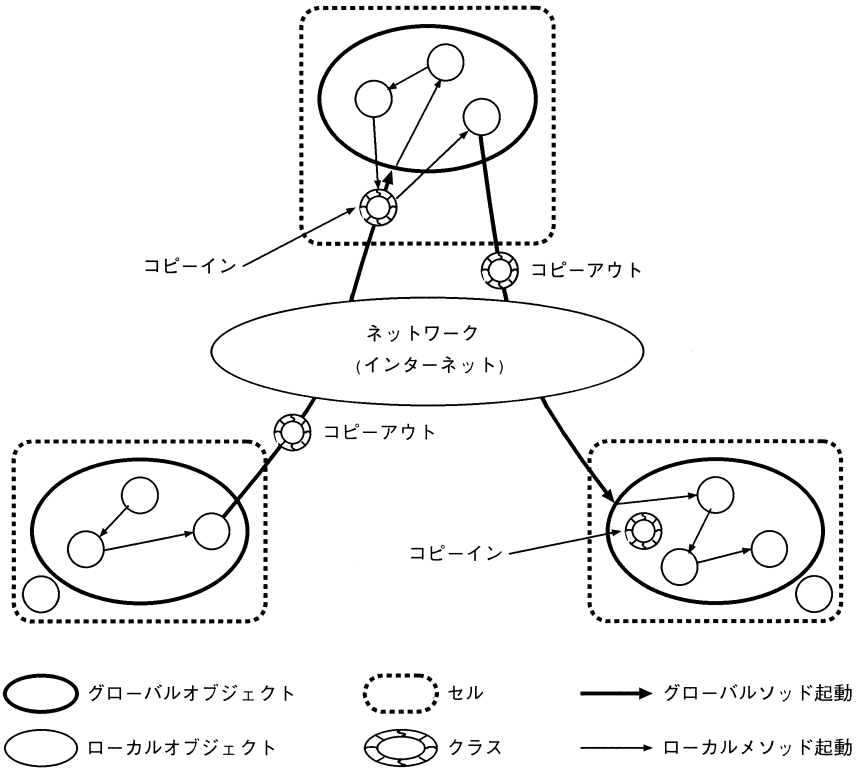

図 1　OZ のオブジェクトモデル

2.1.1 グローバルオブジェクト

ネットワークに対してサービスを提供する窓口となり，インタフェースの役割を果たすオブジェクトが図 1 に示したグローバルオブジェクトである．ネットワーク側からこのオブジェクトのメソッドを呼び出すことができる．従って，このオブジェクトはシステム全体で一意の識別子を持つ．また，ネットワーク側からこのオブジェクト

のメソッドを呼び出すと，その呼び出し毎に別スレッドでメソッドが実行される．このオブジェクトへのメソッド呼び出しを，グローバルメソッド起動と呼ぶ．

2.1.2 ローカルオブジェクト

ネットワークを移動することが可能なオブジェクトが図1に示したローカルオブジェクトである．ネットワーク側からこのオブジェクトのメソッドを呼び出すことはできない．このオブジェクトをグローバルメソッド起動の引数や戻り値に指定すると，オブジェクトのコピーアウト/コピーインが行われ，そのオブジェクトの複製が相手側に渡る．その結果として，このオブジェクトはグローバルオブジェクト間を移動することができる．一方，このオブジェクトへのメソッド呼び出しでは，引数や戻り値に指定されたオブジェクトの複製ではなく,そのオブジェクトの参照が相手側に渡る．このため，グローバルメソッド起動に較べて高速である．グローバルメソッド起動と区別するために，ローカルオブジェクトへのメソッド呼び出しをローカルメソッド起動と呼ぶ．

2.1.3 セ　　ル

一つのグローバルオブジェクトとそれから再帰的に参照されているローカルオブジェクトの集まりが，図1に示したセルである．同じセル内のオブジェクトは，その参照を得て相互にローカルメソッド起動を行うことができる．言い換えれば，メソッド起動の引数や戻り値に指定されたオブジェクトの複製が行われるかどうかの境界がセルであるとも言える．オブジェクトの複製は，そのオブジェクトが参照するオブジェクトに対しても再帰的に複製を行うディープコピーと呼ばれる方法による．従って，ローカルオブジェクトは一つのセルにのみ属し，複数のセルに属することはない．また，セルはグローバルオブジェクトの永続化や実行（スレッド）の管理などの単位としての役割を持つ．

2.2 クラス配送

オブジェクトの移動先にクラスが存在しないとそのオブジェクトを稼働させることができない．OZ は，そのオブジェクトが稼働するために必要なクラスを配送する仕組みを持つ．つまり，拡張や改良したソフトウェアを部品としてネットワーク環境で流通させ，分散システム間で相互に依存する部分を共有化することができる．

OZ は，オブジェクトが稼働する過程で必要とされるクラスが未だ配送されていなければ，そのクラスを供給できるシステムをネットワーク上で検索し，そのシステムからクラスをダウンロードする．このクラスの検索と配送のシステム自身は，OZ を使って構築されており，それ自身が一つの分散システムとなっている．

しかし，配送されてきた未知のクラスを実行することになり，後述するようなセキュリティ上の問題が発生する．

2.3 OZ 言語

OZ では，OZ のオブジェクトモデルに基づくプログラミングのために，専用の言語（OZ 言語）を採用している．OZ 言語は，OZ コンパイラにより実装言語に変換される．OZ 言語は，OZ のオブジェクトとして共通の性質を持つクラスを暗黙に継承するオブジェクト指向言語である．OZ コンパイラは，OZ 言語を実装言語に変換する際にグローバルオブジェクトに対するメソッド起動を行うためのクラスの生成やセ

キュリティに関するコードの埋め込みを行う.

3. セキュリティ

OZ のオブジェクトモデルでは，セル内の範囲とセル外，すなわち，ネットワークを介するセル間とでは，セキュリティに対する要求は異なる．セル間では，様々なセキュリティのレベルが要求されるため，それらに対応できるような柔軟性が重要である．一方，セル内では，配送されてきた未知のクラスを実行するため，柔軟性よりも高速性が重要となる．本稿では，このセル内のセキュリティに絞って述べる.

なお，セル間は認証にもとづくアクセス制御によりセキュリティを確保する．OZ では, セル間の通信であるグローバルメソッド起動の過程でのデータは, 暗号化通信[3] により送受信される. また, 認証とアクセス制御に必要なライブラリが用意されている.

3.1 セル内セキュリティの確保

オブジェクトの移動やクラスの配送がないシステムでは，オブジェクトの振る舞いは，そのシステムの開発者により限定されている．つまり，システムの開発者は，オブジェクトがセキュリティ上問題のある振る舞いをしないようにシステムを作成することができる.

しかし，クラスの配送を伴うシステムでは，システムの開発者は，オブジェクトが稼動するまで，そのクラスにセキュリティ上問題のある振る舞いをするコードが含まれているかを知ることはできない．更に，オブジェクトの移動により，外部からもたらされたオブジェクト（外来オブジェクト）は，実際にはそのシステムの開発者が作成したクラスを使って作成されていない可能性もある．つまり，ありえない状態の外来オブジェクトが稼働することにより，セキュリティ上問題のある振る舞いをする.

セル内セキュリティを確保するために問題となるアクセスを図 2 に示す.

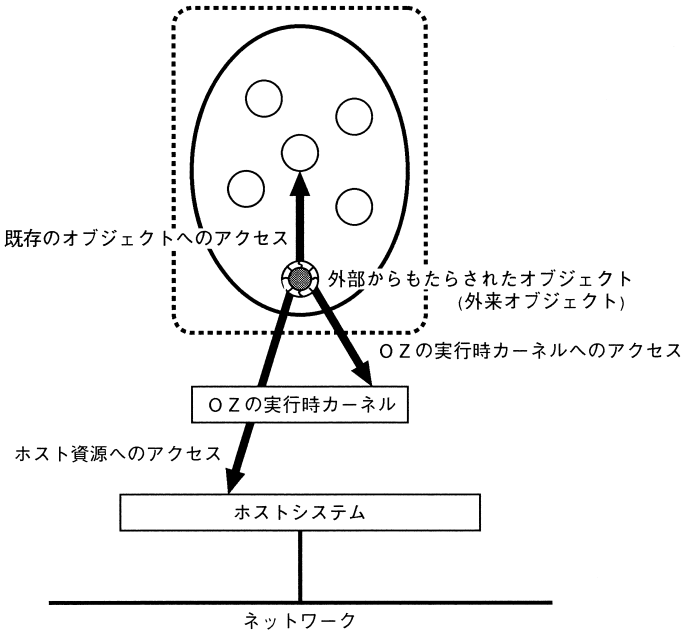

図 2　問題となるアクセス

3.1.1 既存のオブジェクトへのアクセス

オブジェクト指向を採用するOZでは，オブジェクトが持つ情報へのアクセスは，オブジェクトに対してメソッドを呼び出すことにより行う．しかし，外来オブジェクトのメソッドを呼び出すことは，配送された未知のクラスのコードの実行を意味する．つまり，外来オブジェクトのメソッドを呼び出すと，それを手がかりに既存のオブジェクトに対してメソッド起動が行なわれ，既存のオブジェクトが持つ重要な情報にアクセスされる危険がある．

外来オブジェクトは配送されたクラスを使って作成されているため，外来オブジェクトから行われるメソッドの呼び出し自体を禁止することはできない．従って，メソッドが呼び出される側のオブジェクトでそのメソッド起動が外来オブジェクトから行われたものか否かを検査できる必要がある．

3.1.2 OZの実行時カーネルへのアクセス

OZのオブジェクトが稼働するために，OZの実行時カーネルはランタイム時に必要となる基本サービスを提供する．OZのオブジェクトは，これらのサービスを介してOZのシステムが持つ情報にアクセスする．外来オブジェクトが，これらのサービスを介して重要な情報にアクセスする危険がある．このため，外来オブジェクトに対して，OZの実行時カーネルへのアクセスを制限する必要がある．

OZでは，グローバルオブジェクトへのメソッド起動により外来オブジェクトが運ばれる．このグローバルオブジェクトは，何らかのフレームワークに従って作成されており，グローバルメソッド起動で運ばれて来るオブジェクトは，そのフレームワークを前提として稼働する．つまり，外来オブジェクトが必要とするサービスは，移動先であるグローバルオブジェクトが提供する．従って，外来オブジェクトに対してOZの実行時カーネルへのアクセス制限を一意に行っても問題はない．

3.1.3 ホスト資源へのアクセス

ホスト資源へのアクセスは，ホスト資源を抽象化したオブジェクトのメソッドを呼び出すことにより行う．外来オブジェクトは，このオブジェクトのメソッドを呼び出すことにより，ホスト資源にアクセスし，結果として，ホストシステムが持つ重要な情報にアクセスする危険がある．このため，外来オブジェクトに対して，ホスト資源へのアクセスを制限する必要がある．

前節で述べたように，必要なサービスはグローバルオブジェクトにより提供される．従って，外来オブジェクトに対してホスト資源へのアクセス制限を一意に行っても問題はない．

3.1.4 OZのオブジェクトへの成り済まし

OZは，OZ言語で記述されたプログラムを実行することを前提としている．しかし，OZの実装に使用したシステムの実行コードを直接使って，あたかもOZのオブジェクトであるかのような動作をする外来オブジェクトが運ばれてくる可能性がある．

このような外来オブジェクトは，OZの実装に使用したシステムに直接アクセスすることにより，重要な情報にアクセスする危険がある．従って，OZ言語で記述されたオブジェクトか否かを明確に区別し，先述のアクセスを制御できる仕組みが必要となる．

3.2 セル内セキュリティモデル

これまで述べてきたように，外来オブジェクトの稼働にはセキュリティ上の危険が伴う．従って，セキュリティを確保するために，外来オブジェクトの振る舞いに制限を加える必要がある．セル内では高速性が重要であるため，セル間セキュリティのような認証によるアクセス制御は実行効率が問題となる．そこで，オブジェクトに赤と緑の色を付けて次のように区別した．

3.2.1 赤オブジェクト

外来オブジェクトは，外部で開発されたクラスを使って生成されたものとそうでないものとの二つに分けることができる．前者のクラスによる外来オブジェクトの振る舞いは未知であるため，先述のアクセスを制限する必要がある．一方，後者のクラスによる外来オブジェクトの振る舞いは既知ではあるが，セキュリティホールを抱えていたり，あるいは，ありえない状態の外来オブジェクトが運ばれてくる可能性がある．従って，同様に先述のアクセスを制限する必要がある．

システムに危害を加える可能性がある危険なオブジェクトという意味で，このオブジェクトに赤色を付けて区別する．

3.2.2 緑オブジェクト

もとから存在していたオブジェクトは，OZ のクラスライブラリなどの出処が明らかなクラスを使って作成されており，その振る舞いは既知であると考えることができる．また，これらのオブジェクトは，重要な情報を持ち，セキュリティを確保する役割も担っている．従って，OZ のシステムはこのオブジェクトを保護する必要がある．これらの意味で，このオブジェクトに緑色を付けて区別する．

4. セル内セキュリティの実装

OZ では，オブジェクト移動やクラス配送を行うため，作成したオブジェクトやクラスが異なるハードウェア上で動作できる必要がある．また，3.1 節で述べた問題に対処できる必要がある．これらの条件から，OZ を実装するシステムとして Java を選択した．

Java[4]を使って，OZ のシステムがどのようにしてオブジェクトの色を保護し，その一貫性を確保しているか，また，情報を保護しているか，その実現について述べる．

4.1 クラス OzSecure の継承

OZ 言語で記述されたクラスは，OZ コンパイラによりクラス OzSecure（付録 A）を継承した Java のクラスとして実装される．クラス OzSecure は Java 言語で記述されたクラスである．クラス OzSecure は属性として色（color）を持ち，これにより OZ のオブジェクトを赤オブジェクトと緑オブジェクトの二つに区別する．また，配送されてくるクラスは，必ずしもクラス OzSecure を継承しているとは限らないので，クラス OzSecure を継承しないクラスのオブジェクトは無条件に赤オブジェクトとして扱う．

クラス OzSecure には，表 1 に挙げる色（color）に関連するメソッドが定義されている．緑オブジェクトは信頼できるオブジェクトであると同時にその信頼性の一貫性を確保する役割も持つ．従って，緑オブジェクトはオブジェクトの色を変更すること

ができる．オブジェクトの色を変更する際，対象となるオブジェクトの色のみが変更される．そのオブジェクトが参照しているオブジェクトの色までは変更されない．

オブジェクトの色の一貫性を確保するために，赤オブジェクトからはこれらの操作を行うことはできない．ただし，緑オブジェクトも赤オブジェクトも自分自身のオブジェクトやスレッドの色を参照することはできる．Java 言語の package によるアクセス保護機構を使って，表 1 に挙げたメソッドを呼び出す方法以外でオブジェクトの色を変更したり，あるいは，それらのメソッドそのものを再定義したりできないように実装している．

表 1　クラス OzSecure の主なメソッド

メソッド名	概要
changeObjectToGreen	自分自身が緑オブジェクトなら、指定したオブジェクトの色を緑にする。
changeObjectToRed	自分自身が緑オブジェクトなら、指定したオブジェクトの色を赤にする。
changeRunningToGreen	自分自身が緑オブジェクトなら、スレッドの色を緑にする。
changeRunningToRed	自分自身が緑オブジェクトなら、スレッドの色を赤にする。
changeThreadRedIfNecessary	指定したオブジェクトの色が赤なら、スレッドの色を赤にする。スレッドの色を変更した場合は、真を返す。スレッドの色が既に赤の場合は、偽を返す。
checkSecureInvocation	自分自身が緑オブジェクトで、スレッドの色が赤ならばセキュリティ例外を発生する。
isGreen	自分自身のオブジェクトの色が緑なら、真を返す。
isRed	自分自身のオブジェクトの色が赤なら、真を返す。
runningIsGreen	メソッドを実行しているスレッドの色が緑なら、真を返す。
runningIsRed	メソッドを実行しているスレッドの色が赤なら、真を返す。

4.2　スレッドによる色の伝播

赤オブジェクトが行うメソッド起動には危険が伴うものとして扱うが，更に，赤オブジェクトのメソッドを実行する過程で生成されるオブジェクトにも同様の扱いが必要となる．実行中のメソッドが赤オブジェクトに起因するものかを調べるには，スタック上のオブジェクトとメソッドの実行履歴を解析する方法がある．しかし，メソッド起動がネストするに伴って，スタックは深くなり，スタックの解析に時間がかかってしまう．OZ では，スレッドにも色を付けてオブジェクトの色を伝播する方法を取った．スレッドの色であれば，メソッドが呼び出された側で容易に調べることができる．

OZ で実行されるスレッドは，Java のスレッドのクラス java.lang.Thread を継承したクラス ExThread（付録 B）のオブジェクトとして実装している．クラス ExThread は，属性としてクラス OzSecure と同様に色（color）を持つ．スレッドの色は，OZ の実行時カーネルと OZ コンパイラが出力するコード（付録 C）によって操作される．スレッドの色の操作と伝播は次の時に行われる．

1）　赤オブジェクトに対するメソッド起動

緑オブジェクトが赤オブジェクトのメソッドを呼び出す場合，スレッドの色を

赤に変更した上でそのメソッドを呼び出す. そのメソッドが終了すると緑に戻す. この様子を図 3 に示す.

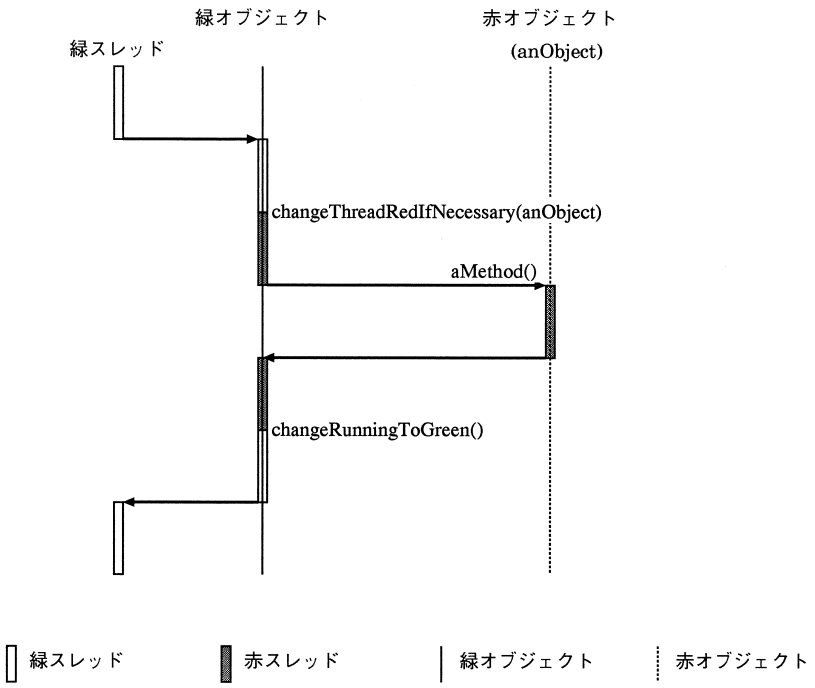

図 3　赤オブジェクトへのメソッド起動

2)　新たなオブジェクト生成

新たに生成されるオブジェクトには，スレッドの色と同じ色を付ける.

3)　新たなスレッド生成

新たに生成されるスレッドには，現在のスレッドの色と同じ色を付ける.

クラス ExThread のオブジェクトでないスレッドの色は， 無条件に赤として扱う. 緑オブジェクトはスレッドの色は変更できるが，赤オブジェクトによってスレッドの色は変更できないので，配送されてきた未知のクラスを実行してもスレッドの色を保護することができる.

4.3　緑オブジェクトの保護

緑オブジェクトは信頼できるオブジェクトであると同時に，重要な情報を保持しているオブジェクトでもある. 従って, 赤オブジェクトからのメソッド起動を禁止する. OZ コンパイラは，OZ 言語で記述されたクラスのメソッドを Java 言語に変換する際に，メソッドの最初に実行されるコードとして，次の二つの条件が成立すると例外（java.lang.SecurityException）を発生するコード（付録 D）を組み込む.

・メソッドを実行しているスレッドの色が赤である.

・メソッドが呼び出されたオブジェクト自身の色が緑である.

これによって, 赤オブジェクトから緑オブジェクトへのメソッド起動を禁止している. この様子を図 4 に示す.

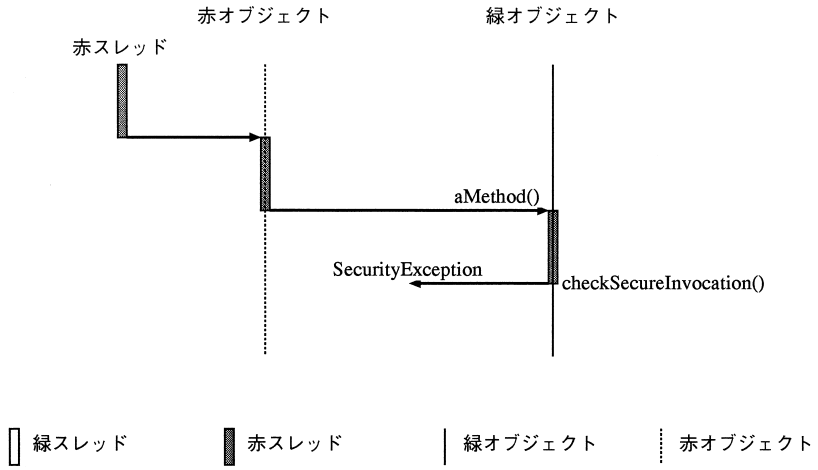

図 4　緑オブジェクトへのメソッド起動

4.4　オブジェクトのコピーアウト/コピーイン

グローバルメソッド起動を行うと，引数や戻り値として指定されたオブジェクトのコピーアウト（オブジェクトの書き出し）やコピーイン（オブジェクトの読み込み）が行われる．赤オブジェクトからもグローバルメソッド起動を行うことが可能なため，グローバルメソッド起動を使って緑オブジェクトが持ち出される危険がある．このためスレッドの色が赤の場合，緑オブジェクトのコピーアウト/コピーインを禁止している．また，コピーインされてくるオブジェクトには，それから再帰的に参照されるオブジェクトも含めて，全てのオブジェクトに一意に赤色を付けている．これによって，外来オブジェクトに赤色を付け，不正に操作された状態のオブジェクトにも対処することができる．

4.5　ロードするクラスの制限

オブジェクトやスレッドの色を不当に操作されないように，Java のクラスローダ（java.lang.ClassLoader）の仕組みを使って，ロードするクラスに次の制限を加えている．

1）　メソッド readObject/writeObject を定義したクラスのロード禁止

これらのメソッドが再定義されていると，オブジェクトのコピーアウト/コピーイン時に，オブジェクトの色が変更される危険がある．

2）　ネイティブメソッドを持つクラスのロード禁止

ネイティブコードによりメモリが直接操作され，オブジェクトの色が変更される危険がある．

3）　スタティックイニシャライザを持つクラスのロード禁止

クラスのロードは，OZ の実行時カーネル内部の特権スレッドで行われる．そして，クラスのスタティックイニシャライザ部分のコードがそのスレッドで実行される．このために，オブジェクトやスレッドの色を変更する，あるいは，セキュリティの仕組みを無効にするコードがスタティックイニシャライザ部分に組み込まれ，セル内セキュリティの仕組みが働かない状態で実行される危険がある．

これらの制限によって，未然にオブジェクトやスレッドの色を不正に操作するクラスを排除している．

4.6 OZ の実行時カーネルとホスト資源へのアクセス制御

Java のセキュリティマネージャ（java.lang.SecurityManager）の仕組みを使って，スレッドの色を判定の基準としてホスト資源へのアクセスを制御する．同様に，OZ の実行時カーネルの各サービスでもセキュリティマネージャを呼び出す．これらのアクセス制御の概要を表 2 に示す．このようにして，赤オブジェクトに起因する OZ の実行時カーネルとホスト資源へのアクセスを制御している．

表 2　スレッドの色によるアクセス制御

	スレッドの色	
操作	緑	赤
OZ のシステムのファイルの読み込み	できる	特定のディレクトリ下のみ
OZ のシステムのファイルの書き込み	できる	特定のディレクトリ下のみ
OZ のシステムのファイルの削除	できる	特定のディレクトリ下のみ
ホストのファイルの読み込み	できる	できない
ホストのファイルの書き込み	できない	できない
ホストのファイルの削除	できない	できない
ソケットの操作	できる	できない
外部コマンドの実行	許可されたもののみ	できない
スレッドの制御	できる	できない
クラスローダの作成	できない	できない
プロパティのアクセス	できる	できない
セルの操作	できる	できない
クラスファイルの読み込み	できる	公開されたもののみ
クラスファイルの書き込み	できる	できない
公開クラスの操作	できる	できない

5. AWT のイベント関連クラスのラッピング

OZ のセル内セキュリティモデルによるプログラミングで，Java で提供されているクラスを利用できるようにするために，それらをクラス OzSecure を継承するクラスにラッピングしている．

Java の AWT[*3] の実装では，マウスのクリックなどのイベントが発生すると，AWT 内部で生成されたスレッド（AWT スレッド）上でユーザ定義のオブジェクトのメソッドを呼び出すコールバック形式になっている．このユーザ定義のオブジェクトが緑オブジェクトであると，AWT スレッド上で緑オブジェクトのメソッドを呼び出すことになる．

しかし，OZ ではクラス ExThread により生成されていないスレッドを一意に赤スレッドとして扱うため，AWT スレッド上での緑オブジェクトへのメソッド起動はセキュリティ例外となる．

従って，AWT のイベント関連のクラスをラッピングする際に，赤スレッド上で緑オブジェクトのメソッドを呼び出す工夫が必要となる．

5.1 AWT スレッド上での緑オブジェクトへのメソッド起動

Java の AWT 関連クラスは，OZ のシステムが稼動の前提としている Java の一部であることから，認証されたクラスであると言って良い．また，AWT スレッド上での緑オブジェクトへのメソッド起動は，外来オブジェクトからのメソッド呼び出しとは異なり，ユーザ自身に起因するものである．つまり，このメソッド起動は，見掛け上は緑オブジェクトから行われたメソッドの呼び出しと見なして良い．従って，AWT スレッド上での緑オブジェクトへのメソッド起動時に発生するセキュリティ例外は，一意に回避しても良い．

5.2 待ち行列を介した緑オブジェクトへのメソッド起動

OZ のセル内セキュリティモデルでは，AWT スレッド（赤）上で緑オブジェクトのメソッドを直接に呼び出すことはできない．そこで，図 5 のように待ち行列を介して間接的に緑オブジェクトのメソッドを呼び出す．

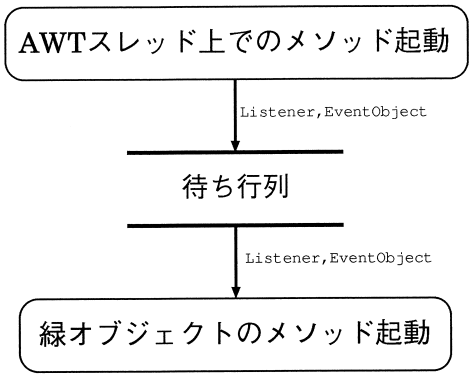

図 5　待ち行列を介した緑オブジェクトへのメソッド起動

AWT のイベントが発生すると，AWT スレッド上で仲介役のオブジェクト（adapter）のメソッドが呼び出される．その仲介役のオブジェクトは，対応するユーザ定義の緑オブジェクト（listener）とイベント情報（EventObject）を待ち行列に追加する．一方，あらかじめ生成しておいたスレッド（緑）上でその待ち行列からその二つの情報を取出し，AWT スレッドに代ってユーザ定義の緑オブジェクトのメソッドを呼び出す．このようにして，見掛け上は AWT スレッド上から緑オブジェクトのメソッドを呼び出しているように見える．

主要部分のクラス概要とクラス図をそれぞれ表 3 と図 6 に示す．また，メソッドの呼び出しの様子を図 7 に示す．但し，これらは説明を簡単にするためのもので，実際の実装とは異なる．

表 3　主要部分のクラス概要

クラス名	概要
OzEvent	メソッド起動を受ける緑オブジェクトと AWT からのコールバック時に渡されるクラス EventObject を継承したオブジェクトの組を保持する。また、このクラスを使って生成されたオブジェクトが待ち行列の要素の値になる。クラス OzSecure を継承する。
OzEventQueueSlot	待ち行列の要素である。クラス OzEvent を使って生成されたオブジェクトを値として保持する。クラス OzSecure を継承する。
OzEventQueue	赤スレッドからの緑オブジェクトへのメソッド起動の要求の待ち行列である。クラス OzSecure を継承する。
OzEventDispatch	クラス OzEventQueue のオブジェクトを作成すると、このクラス OzEventDispatch のオブジェクトが作成される。また、同時に新しいスレッドが生成され、メソッド run が呼び出される。メソッド run は、待ち行列から要素を取出し、クラス OzEventListener を継承した緑オブジェクトのメソッド dispatch を呼び出すという処理を繰り返す。クラス OzSecure を継承する。
OzEventListener	クラス OzEventDispatch からメソッドを呼び出すためのインターフェスである。メソッド起動を受ける緑オブジェクトはこのクラスを継承する必要がある。クラス OzSecure を継承する。
OzActionListener	AWT スレッドに代ってクラス OzEventDispatch で作成されたスレッド上でメソッド dispatch が実行される。このメソッドが、ユーザ定義のメソッド actionPerformed を呼び出す。クラス OzSecure を継承する。
ActionListenerAdapter	AWT スレッド上でコールバックを直接受けるメソッド actionPerformed が定義されている。このメソッドは、待ち行列に緑オブジェクトのメソッド起動に必要な情報を登録する。クラス OzSecure を継承しない。

6. プログラミング上の注意

OZ のセル内セキュリティモデルは，常に緑オブジェクトを保護するが，このことはプログラミング上の制約にもなってくる．つまり，プログラム設計の段階からオブジェクトの色を考慮する必要がある．これは OZ のセル内セキュリティモデルの長所でもあり，同時に短所でもある．このセキュリティを有効なものとするために，次の点に注意しなければならない．

1)　安易にオブジェクトの色を緑に変更しない

緑オブジェクトは常に安全であるという前提が破綻し，もはやオブジェクトの色によるセキュリティが無意味となってしまう．この操作を行う場合，オブジェクトの認証やクラスの出処確認などが必須である．

2)　常に赤オブジェクトは保護されない

赤オブジェクトは常に他の赤オブジェクトによってアクセスされ，外部に持ち出される危険がある．つまり，オブジェクトの色を赤に変更すると，そのオブジェクトが直接に持つ情報はもはや保護されない．重要な情報を持つ緑オブジェクトの色を変更する際には注意が必要である．

7. 考　　察

OZ は，統一されたモデルによる共通環境を提供し，オブジェクトの移働に伴って

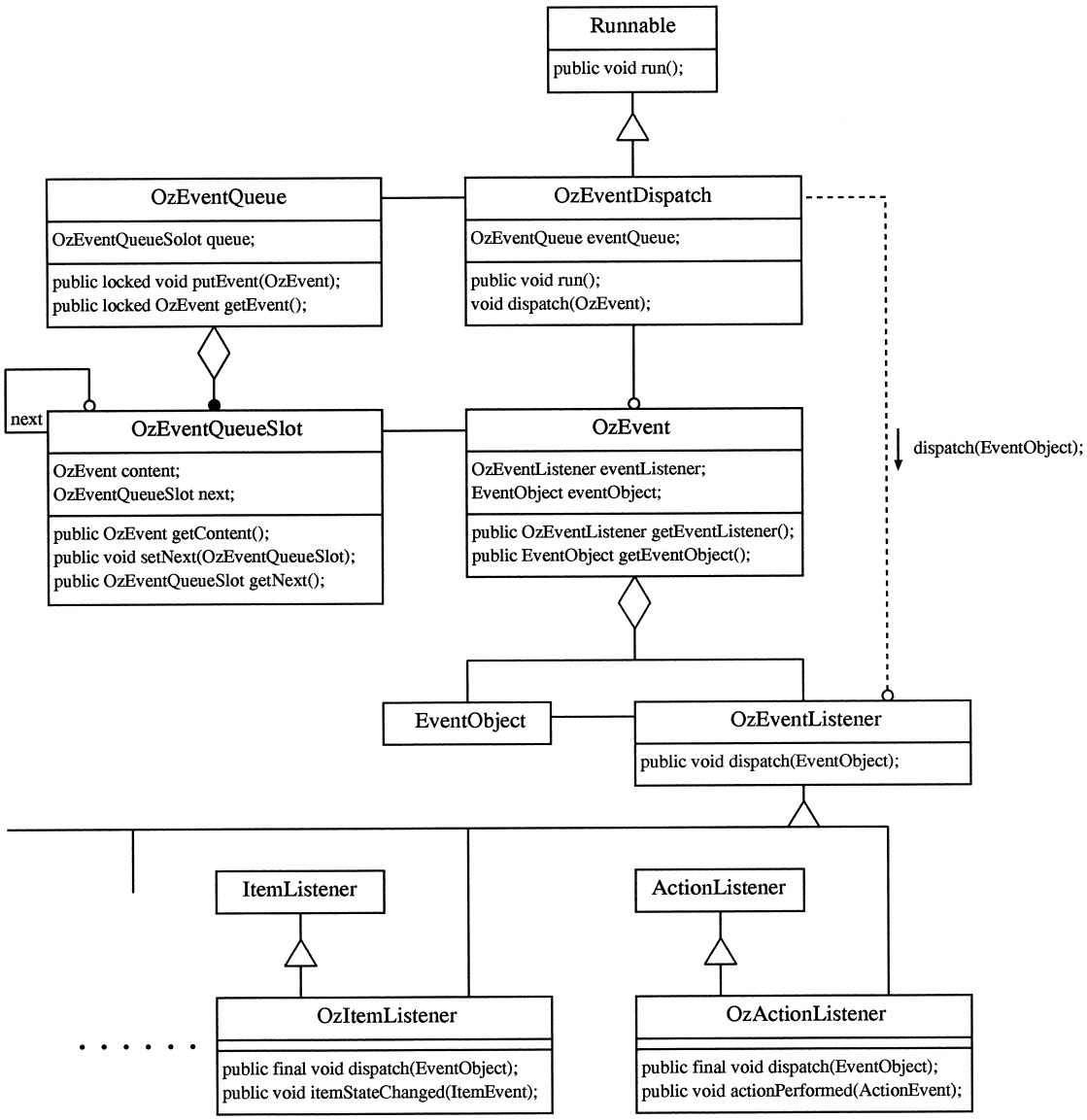

図 6　主要部分のクラス図

その稼働に必要なクラスを配送する．これによって，分散システムの開発時に必要なソフトウェアの共有と稼働時に必要なソフトウェアの流通を実現している．

従来の分散システムでは，開発時や稼働時に必要なソフトウェアをあらかじめ準備する必要があった．しかし，OZ が提供する環境では，ソフトウェアの共有と流通が自動化されたことにより，必要なソフトウェアをあらかじめ準備する必要がない．すなわち，分散システムのフレームワークが構築できれば，段階的にシステムを実装したり，あるいは，後からその拡張や改良ができる．

システムに柔軟性を与える技術として，戸叶[5]が述べているようなエージェントが注目されている．OZ 自身は，そのようなエージェントを想定したシステムではない

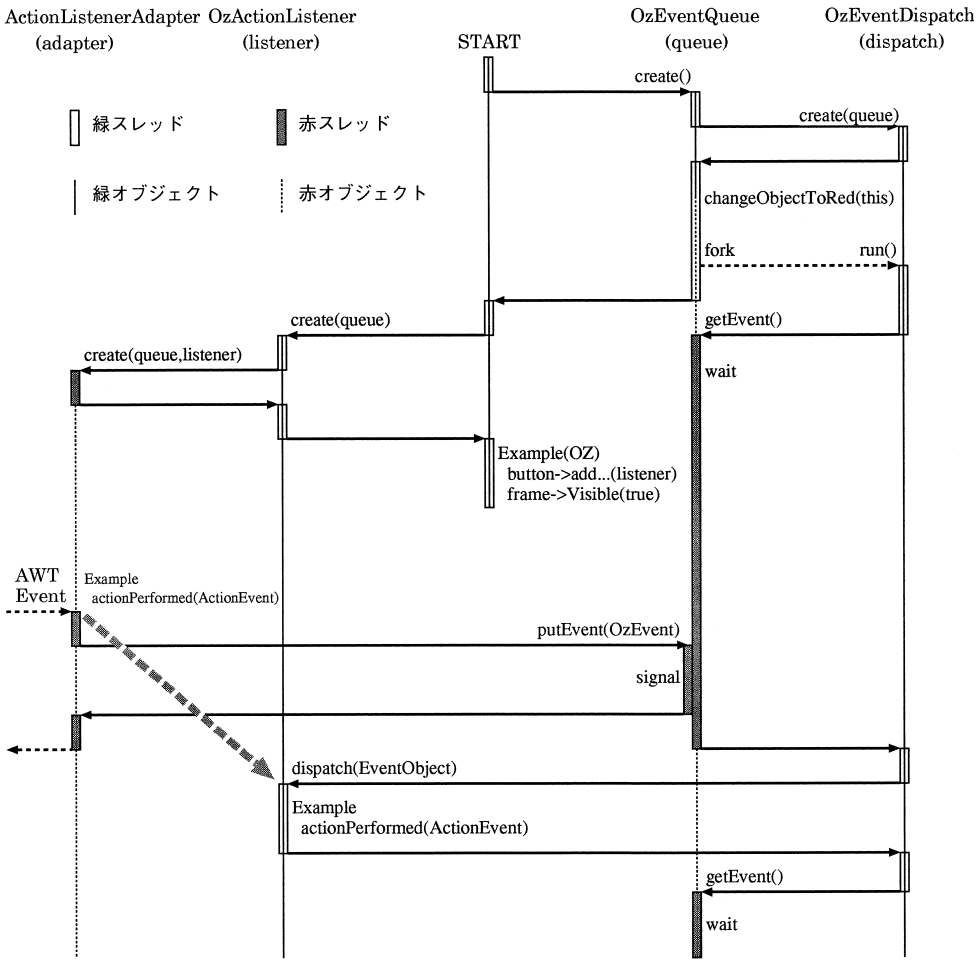

図 7　AWT コールバックと緑オブジェクトへのメソッド起動

が，エージェントシステムを構築するための環境として十分な機能を持っていると考える．そして，エージェントシステムにおける課題の一つであるセキュリティ問題について，その解決のための基本となるセキュリティ確保の仕組みを提供している．

8. おわりに

移動してきたオブジェクトに色を付け，その色によりオブジェクトの振る舞いに制限をかける．これによって配送されてきた未知のクラスをセキュリティを確保しながら実行することができる．この仕組みと Java による実現について述べた．

OZ のシステムは，クラスが配送されることがないシステムに較べて，セキュリティ上の問題が発生する可能性が高いが，これまで述べてきた仕組みによりセキュリティを確保することが可能になる．そして，この仕組みを使って，個々の分散システムがその運用方針に従ったセキュリティを構築することができる．

現在，OZ のシステムはそのソースコードとともに公開[6]されている．

謝　辞　本稿執筆にあたり貴重なコメントをいただいた西岡利博氏（三菱総合研究所研究員）に感謝する．

本研究は，情報処理振興事業協会「創造的ソフトウェア育成事業」の一環として行われたものである．

＊1　Request for Comments：インターネットで使うプロトコルやサービスなどについて記述した一連の文書．

＊2　Object-Oriented Distributed Systems Environment の頭文字 OODSE の発音から．OZ の開発は，1986 年から実施された「分散オブジェクト指向言語システムの研究開発」で作成された初代 OZ から始まり，OZ＋，OZ＋＋［1］を経て OZ＋＋＋［2］（現在の OZ）に至っている．筆者は，OZ＋＋開発の後半（1994 年度）から OZ プロジェクトに参加した．

＊3　Abstract Window Toolkit：グラフィカルユーザインタフェースを作成するためのアプリケーションインタフェース．

参考文献

［1］　塚本享治, オブジェクト指向分散処理環境（OZ＋＋）の研究開発, 開放型基盤ソフトウェア研究開発評価事業報告書, 情報処理振興事業協会, 1996 年 3 月.

［2］　塚本享治, 西岡利博, 濱崎陽一, OZ：セキュアなワールドワイドオブジェクト指向分散環境の開発, 創造的ソフトウェア育成事業及びエレクトロニック・コマース推進事業最終成果発表会論文集, 創造的ソフトウェア育成事業編, 情報処理振興事業協会, 1998 年 5 月, pp. 119～126.

［3］　濱崎陽一, 樋口忠幸, 西岡利博, 塚本享治, オブジェクト指向分散環境 OZ の暗号化通信プロトコル, 情処研報, Vol. 97, No. 77, 情報処理学会, 1997 年 5 月, pp. 19～24.

［4］　Sun Microsystems Inc., Java Development Kit Version 1.1.4.

［5］　戸叶徹, エージェント――柔軟なシステムを目指して, 技報通巻 57, Vol. 18, No. 1, 日本ユニシス(株), 1998 年 5 月, pp. 69～95.

［6］　http : //www.etl.go.jp/etl/bunsan/OZ_Proj/

［7］　Yoichi Hamazaki, Toshihiro Nishioka and Michiharu Tsukamoto, The Security Mechanism of OZ : an Object-Oriented Distributed Systems Environment, Proceedings of Object-Based Parallel and Distributed Computation France-Japan Workshop, OBPDC '97 Toulouse, France.

［8］　Jan Vitek, Manuel Serrano and Dimitri Thanos, Security and Communication in Mobile Object Systems, Lecture Notes in Computer Science, Vol. 1222, Springer-Verlag, 1997, pp. 176～199.

［9］　Giovanni Vigna (Ed.), Mobile Agents and Security, Lecture Notes in Computer Science, Vol. 1419, Springer-Verlag, 1998.

［10］　西岡利博, 濱崎陽一, 塚本享治, オブジェクト指向分散環境 OZ のセキュリティモデル, 情処研報, Vol. 97, No. 78, 情報処理学会, 1997 年 8 月, pp. 103～108.

［11］　濱崎陽一, 西岡利博, 塚本享治, オブジェクト指向分散環境 OZ のプロセス内セキュリティ, 情報処理学会第 54 回（平成 9 年前期）全国大会, 講演論文集（1）1 K-7, 情報処理学会, 1997 年 3 月, pp. 297～298.

［12］　佐藤一朗, モバイルエージェントの動向, 人工知能学会誌, Vol. 14, No. 4, 人工知能学会, 1999 年 9 月, pp. 22～29.

付録

A　クラス OzSecure のプログラム概要

クラス OzSecure の色に関係する部分のプログラムの概要を示す．全ての OZ のオブジェクトは，クラス OzSecure を継承することにより色を持つ．クラス OzSecure は，その色の保護を行うようにプログラムされている．なお，クラス OzMonitor は OZ のノンリエントラントなモニタ機能を実現するためのクラスである．

```
package JP.go.ipa.oz.system;
// ...

public
class OzSecure extends OzMonitor implements Serializable
{
    // ...
    public final static boolean GREEN = true;
    public final static boolean RED = false;
    boolean color;
    // ...
    protected
    OzSecure()
    {
        Thread t = Thread.currentThread();
        if (t instanceof ExThread) {
            color = ((ExThread)t).color;
        } else {
            color = RED;
        }
    }
    // ...
    protected final
    void checkSecureInvocation()
    throws SecurityException
    {
        Thread t = Thread.currentThread();
        if (t instaceof ExThread) {
            if(! ((ExThread)t).color && color) {
                throw new SecurityException("...");
            }
        } else if (color) {
            throw new SecurityException("...");
        }
    }
    // ...
    protected final
    boolean changeThreadRedIfNecessary(Object callee)
    {
        Thread t = Thread.currentThread();
        if ((t instanceof OzSecure)
                && (((OzSecure)callee).color == GREEN)) {
            if ((callee instanceof OzSecure)
```

```
                    && (((OzSecure)callee).color == GREEN)) {
                return false;
            } else {
                ((ExThread)t).color = RED;
                return true;
            }
        } else {
            return false;
        }
    }
    // ...
    protected final
    void changeRunningToGreen()
    throws SecurityException
    {
        if (color) {
            Thread t = Thread.currentThread();
            if (t instance of ExThread) {
                ((ExThread)t).color = GREEN;
            } else {
                throw new SecurityException("...");
            }
        } else {
            throw new SecurityException("...");
        }
    }
    // ...
}
```

B　クラス ExThread のプログラム概要

クラス ExThread の色に関係する部分のプログラムの概要を示す．クラス ExThread を使って生成されたスレッド上で，OZ のオブジェクトのメソッドが実行される．クラス ExThread はクラス OzSecure と同様の色を持ち，その色の伝播を行うようにプログラムされている．なお，セルは，スレッドを管理するため，スレッドグループとしての役割を持つ．

```
package JP.go.ipa.oz.system;
// ...

final
class ExThread extends Thread
{
    // ...
    public static boolean GREEN = OzSecure.GREEN;
    public static boolean RED = OzSecure.RED;
    boolean color = RED;
    // ...
    private
    void init(Runnable target)
    {
        Thread t = Thread.currentThread();
        if (t instanceof ExThread) {
```

```
            if (target instaceof OzSecure) {
                OzSecure secure = (OzSecure)target;
                if (secure.color == OzSecure.RED) color = RED;
                else color = ((ExThread)t).color;
            } else {
                color = RED;
            }
        } else {
            color = RED;
        }
    }
    // ...
    ExThread(ThreadGroup cell, Runanble target, ExChannel callee)
    {
        super(cell, target);
        init(target);
        // ...
    }
    // ...
}
```

C　OZ コンパイラが出力するメソッド起動の Java プログラム

オブジェクト anObject のメソッド aMethod を起動する OZ 言語のプログラムは，OZ コンパイラにより次のように出力される．

```
// OZ 言語: anObject->aMethod()
if (changeThreadRedIfNecessary(anObject)) {
    try {
        anObject().aMethod();
    } finally {
        changeRunningToGreen();
    }
} else {
    anObject.aMethod();
}
```

D　OZ コンパイラが出力するメソッド定義の Java プログラム

クラス Sample のメソッド aMethod を定義する OZ 言語のプログラムは，OZ コンパイラにより次のように出力される．

```
public
class Sample extends JP.go.ipa.oz.system.OzSecure
{
    // ...
    public void aMethod()
    {
        checkSecureInvocation();
        // ...
    }
    // ...
}
```

執筆者紹介　吉　田　泰　光（Yasumitsu Yoshida）

1966 年 6 月 12 日生．1989 年 3 月上智大学理工学部物理学科卒業．1991 年 1 月日本ユニシス(株)入社．1994 年から 1997 年に渡って情報処理振興事業協会の事業による OZ プロジェクトに参加．現在，ソリューションシステム部ソリューション技術室に所属し，エキスパート構築支援ツールの保守を担当．

Yasumitsu.Yoshida@unisys.co.jp

塚　本　享　治（Michiharu Tsukamoto）

1949 年 9 月 10 日生．1972 年 4 月東京大学工学部計数工学科卒，同年 5 月電子技術総合研究所入所．制御部システム制御研究室・情報制御研究室主任研究官，情報アーキテクチャ部分散システム研究室長，同部総括主任研究官，歴任ののち，1999 年 4 月東京工科大学メディア学部教授．分散システム，ソフトウェアメトリックスの研究を行っている．情報処理学会，ACM，IEEE-CS 会員．

tukamoto@media.teu.ac.jp