プログラミング 14−18
（1997. 8. 22）

オブジェクト指向分散環境 OZ のセキュリティモデル

西岡 利博　　濱崎陽一　　塚本 享治
nishioka@oz.ipa.go.jp　　{hamazaki,tukamoto}@etl.go.jp
三菱総合研究所　　電子技術総合研究所
〒100 千代田区大手町 2-3-6　　〒305 つくば市梅園 1-1-4

オブジェクト指向分散環境 OZ は、他のユーザが開発した未知のクラスのオブジェクトでも、ネットワーク上でやりとりして動作させられることが特長である。このようなシステムでは、ネットワークから輸入したオブジェクトによって、ホストシステムがダメージを受けないことと同時に、既存のオブジェクトにも適切な保護を与えることが重要である。本稿では、このためのセキュリティモデルを説明する。

このセキュリティモデルは、リモートメソッド起動におけるユーザ認証に基づくアクセス制御と、それを通して輸入されたオブジェクトから既存のオブジェクトとホストシステムとを保護する機構とによって成り立っている。このセキュリティモデルにより、輸入されたオブジェクトの挙動は著しく制限されるので、必要に応じて安全に緩和するためのプログラミングテクニックが必要となる。本稿では、このようなプログラミングテクニックについても述べる。

The Security Model of OZ: an Object-Oriented Distributed Systems Environment

Toshihiro Nishioka　　Yoichi Hamazaki　Michiharu Tsukamoto
nishioka@oz.ipa.go.jp　　{hamazaki, tukamoto}@etl.go.jp
Mitsubishi Research Institute　　Electrotechnical Lab.
2-3-6, Otemachi, Chiyoda-ku, Tokyo, 100 Japan　　1-1-4, Umezono, Tsukuba, Ibaraki, 301 Japan

The OZ, an object-oriented distributed systems environment, features that any objects of unknown classes developed in anywhere in the world can be passed over the network. In such systems, it is important to protect the already-existed objects, as well as to protect the host system being damaged from the imported objects. This paper explains the security model to achieve it.

This security model consists of the access control of the remote method invocations using the user authentication and the protection mechanism for the host system and the already-existed objects from the imported objects. Since the behavior of the imported objects are extremely limited by this security policy, in some situations, several programming techniques are required to relax this limitation safely. This paper also describes these programming techniques.

1 はじめに

WWW という限定された環境の上ででも、多くのユーザが、CGI や Java などを駆使して、独自のネットワークサービスを提供している今日のネットワークの状況を見れば、同じことを、より少ない制約と記述量で提供できる環境の登場が望まれるのもそれほど先のことではないに違いない。オブジェクト指向分散環境 OZ は、独自のネットワークサービスを、分散システムとして、安全にネットワーク上に公開できる環境である。OZ のユーザは、新しいサービスを、分散システム記述用のオブジェクト指向言語 OZ のオブジェクトとしてプログラミングして生成する。これを利用するためのインタフェースか、またはクライアントライブラリを公開すれば、世界中の OZ ユーザからリモートメソッド起動 (RMI) による利用が可能となる。ここでの課題は、多くの異なる組織が提供している各種のサービスを、いかに相互運用性を保ったまま独立に内容 / 品質 / インタフェースを拡張 / 変更できるか [1, 2, 3] という点と、そのような環境でいかに安全に運用するか [4] という点である。本稿は後者の課題に関するものである。

相互運用性を保持したままでの拡張 / 変更のためには、ネットワークを越えて未知のオブジェクトを配送して動作させる機構が不可欠である [2]。一方でこの機能はセキュリティ上の問題を生じる可能性がある。本稿では、以下の順序で、OZ の提供するセキュリティモデルと、それに則った安全なプログラムを記述する技術について述べる。

2 OZ のオブジェクトモデル

2.1 セル

OZ では、オブジェクトは “セル” と呼ばれる単位で管理される。セルとは、ひとつの “グローバルオブジェクト” と 0 個以上の “ローカルオブジェクト” からなるオブジェクトの集合である (1)。両者の相違は、セルの外部からはグローバルオブジェクトのメソッドしか起動できないという点である。したがって、グローバルオブジェクトのクラスはセルのインタフェースを規定するので、これを指して “セル” と呼ぶこともある。

セルは分散の単位である。異なるセルに属すオブジェクトは、他の計算機上で動作している可能性がある。RMI の引数や返り値にオブジェクト型が現れる場合、それらはディープコピーされて渡されるので、オブジェクトが複数のセルに属すことはない。セルは

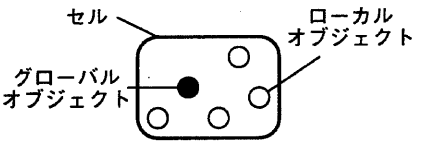

図 1: セル

永続化の単位でもある。セルを永続化すると、そのセルのグローバルオブジェクトから再帰的に参照されている範囲のオブジェクトが、そのセルのオブジェクトとして二次記憶に保存される。

2.2 オブジェクトの実現

セルを動作させるのは “エグゼキュータ” と呼ばれるランタイムカーネルであり、Java で実装されている。エグゼキュータは、セル内のオブジェクトに、メモリ空間、マルチスレッドの実行機構、RMI 機能などを提供する。

永続化されたセルは、“OZ ホーム” と呼ばれるディレクトリ階層の一画に保存される。ひとつのセルにはひとつのディレクトリが与えられ、これを GOD (Global Object Directory) と呼ぶ。

OZ ホームは、“OZ ホーム名” と呼ばれる識別子によって識別される。OZ ホーム名は、多くの場合、DNS のドメイン名と、プラットフォーム OS 上でのユーザ名からなっている。セルは、OZ ホーム名に OZ ホームからの GOD の相対パスを加えたもので識別される。これを GOL と呼ぶ (図 2)。

```
etl.go.jp:hamazaki:games.nethack
```

図 2: GOL の例

3 OZ のセキュリティモデル

OZ のセキュリティモデルは、以下に述べるような二重の構造となっている。

3.1 セル間セキュリティ

セルには重要な情報が含まれている可能性がある。したがって、例えば、オーナ以外のユーザにはセルの内容の改変を許したくないという場合は頻繁に生じる。しかし、クラスのインタフェースはメソッドシグネチャで規定されるので、オーナには改変を許すとすれば、そのためのメソッドが必要であり、オーナ以外のユーザがそのメソッドを起動すれば、改変できてし

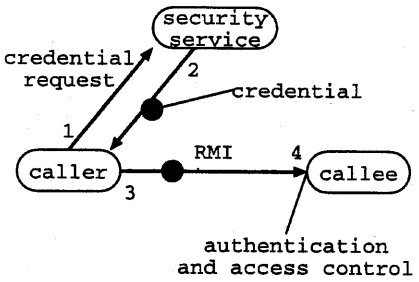

図 3: 認証手続き

まう。そこで、実行時に、そのメソッドを呼び出したユーザが誰であるかを特定し、そのユーザがそのメソッドを起動する権限を持っているかどうかを照合する必要がある。この、caller のユーザを特定する手続きを “認証” と呼ぶ。OZ では、これを以下のように実現する (図 3)。

ユーザ

OZ では OZ ホームをユーザとみなす。すなわち、同じ OZ ホーム上のオブジェクトは、すべて同じユーザの所有物と考える。異なる OZ ホームのオーナは、実際は同一人物であってもシステム上は別のユーザである。ユーザは OZ ホーム名によって識別する。このため、OZ ホーム名を “ユーザ識別子” とも言う。

認証方式

Needham-Shroeder の鍵配送アルゴリズム [5] を用いる。このアルゴリズムでは、認証に必要なクリデンシャルを発行する認証サービスが必要である。OZ における認証サービスとしては OZ のオブジェクトである “セキュリティサービス” が用意されている。caller 側は、callee オブジェクトの GOL からユーザ識別子を抽出し、セキュリティサービスにクリデンシャルの発行を要求する。この手続きはエグゼキュータによって自動的に実行され、クラスを記述する際にはコーディングしない。

アクセス制御

callee 側は、caller から受けとったクリデンシャルから、caller オブジェクトのオーナを特定する。この手続きもエグゼキュータによって自動的に実行される。セルのプログラムは、そうして特定されたユーザ識別子を取り出すことができるので、そのメソッドの起動を許可されているユーザでない場合には、適切な例外を生じるなどして起動を拒否することができる。この、ユーザ識別子の検査は、一般的には、自身のオーナとの比較や、あらかじめ設定されたユーザ識別子の集合に含まれているかどうかの検査などである。この種のプログラミングの支援のために、ACL (Access Control List) ライブラリが提供される。

認証のタイミング

一度クリデンシャルを取得すると、同じユーザとの間では同じクリデンシャルを使用できる。エグゼキュータは、RMI の callee ユーザとクリデンシャルとの対応表を保持し、できるだけ再利用する。ただし、クリデンシャルには有効期限があり、それを越えて使用することはできない。有効期限が切れた場合はセキュリティサービスから再取得する。

3.2 セル内セキュリティ

RMI によってセルの外部からもたらされたオブジェクトは、未知のクラスのインスタンスであるかもしれず、どのような挙動を示すか予測できない。これらの危険なオブジェクトから、重要なオブジェクトおよびホストシステムを保護する必要がある。しかし、セルの中にまで、セル間セキュリティのような認証に基づくアクセス制御を持ち込むと、効率上の問題がある。そこで、以下のような解決方法を提案した [4]。

オブジェクトの色

危険なオブジェクトから重要なオブジェクトに対するメソッド起動を許さないために、次のようにオブジェクトを色分けする。

最初からセル内に存在するオブジェクトは、安全であることを示すために、緑に塗る。緑オブジェクトが生成するオブジェクトは緑オブジェクトである。一方、RMI によってセルの外部からもたらされるオブジェクトは、危険であることを示すために、赤く塗る。赤オブジェクトが生成するオブジェクトもまた赤オブジェクトである。

このように区別し、赤オブジェクトは緑オブジェクトのメソッドを起動できないものとすれば、重要なオブジェクトを保護できる。しかし、OZ を含めて、一般のオブジェクト指向言語では、callee を特定することはできても caller を特定することはできないので、caller が赤であることを確認するためには工夫が必要である。OZ では、スレッドも色分けした。RMI によって発生したスレッドは、最初は緑であり、赤オブジェクトのメソッドを起動するところで、caller 側が

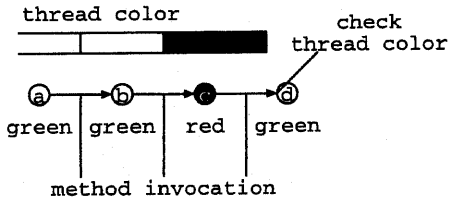

図 4: スレッドの色による危険なオブジェクトの識別

スレッドを赤く塗り替える。スレッドの色を元に戻せるのは緑オブジェクトだけである。こうすることにより、緑オブジェクトは、自分のメソッドを起動したスレッドの色を確認することによって、自分を呼び出しているオブジェクトの色を確認できる。図4の例では、緑オブジェクトdは、緑オブジェクトbがスレッドの色を赤に変更してから赤オブジェクトcを呼び出しているので、スレッドの色を判定することによって赤オブジェクトから呼び出されていることが分かり、メソッド起動は拒否される。

オブジェクトやスレッドの色分けと、それに基づくアクセス制限は、エグゼキュータとコンパイラによって実現され、OZ のプログラマは記述しない。

ホストシステムの保護

ホストシステム上の資源 (ファイル、通信ポート、外部プロセスなど) に対するアクセスは、Java Applet と同様に、一定の方針で制限される。Java Applet では、このためにクライアント側で状態を保存できないという問題が生じたが、赤オブジェクトは、それ自体をセルの一部として保存することが可能であり、問題とはならない。

緑オブジェクトに対しても、ホストシステムに回復不能な破壊が生じる可能性のある挙動 (OZ ホームの外部のファイルの上書き、外部コマンドの実行など) は、制限されたり、許可を求めたりするようになっている。これは、緑オブジェクトといえども完全に信用はできない場合に備えた設計である。そのような状況としては、例えば、開発初期の、セキュリティホールをかかえた状態のプログラムを試験する場合が考えられる。

赤オブジェクトによる RMI

赤オブジェクトによる RMI が、そのオーナの権限で実行されると、同じオーナの所有する他のセルを破壊できてしまう。これを防ぐため、赤オブジェクトによる RMI は、システム定義のユーザ "nobody" の権限で実行される。また、セル内の緑オブジェクトを RMI の引数としてディープコピーしてどこかへ持ち出されないために、赤オブジェクトが緑オブジェクトを引数にして RMI を実行するのは禁止されている。

表 1: セル内外でのセキュリティ方針の差異

	セル間	セル内
行為者の識別	ユーザ認証	赤 / 緑
アクセス制御	プログラムで制御	メソッド起動不可

表1に、セル間セキュリティとセル内セキュリティの差異をまとめた。

4 セルのセキュリティ設計

OZ では、ユーザ認証の機能と赤 / 緑オブジェクトの識別の機能が提供されているが、これらを有効に活用するためには、セルのプログラマが適切なセキュリティ設計を行う必要がある。

多くのセルでは、赤オブジェクトにメソッド起動する必要があるだろう。また、そのうちの多くでは、赤オブジェクトをセルの一部として永続化するかもしれない。しかし、異なるセルから輸入した赤オブジェクトどうしが、セルの内部でメソッド起動し合う必要があるという場合は少数派であろう。以下では、これらの典型的な場合における、セルのセキュリティ設計の指針を示す。

4.1 セルインタフェースの設計

セルのインタフェースは、オーナ以外のユーザにサービスを提供するためのメソッドと、オーナ (またはオーナグループ) だけが起動可能なメソッドを分けて設計する。

サービス提供用のインタフェースは、誰に提供するものであれ、どのような呼ばれ方をしても、セル自身の一貫性やサービス提供能力にダメージを与えられないように設計する。そのような副作用を引き起こせるのはオーナだけに限定すべきである。

オーナだけが起動可能なメソッドでは、起動者がオーナであることを確認する。

4.2 認証によるオブジェクトの色の変更

RMI によって送り込まれるオブジェクトが、オーナだけが起動できるメソッドによって送り込まれるものであって、なおかつ、そのクラスが安全であるとあ

らかじめ分かっている場合は、赤オブジェクトのままにしておく必要はない。緑スレッドは赤オブジェクトを緑オブジェクトに変更できるので、緑オブジェクトに変更すると、設計が簡単になる。

4.3 セル内部での赤オブジェクトの識別

赤オブジェクトに対するメソッド起動は終了しない場合が考えられ、一般にタイムアウトをかけた方がよい。これはそのようにプログラミングする必要があるので、セルの中でどのオブジェクトが赤オブジェクトである可能性があるのかは、明らかでなければならない。RMI の引数や返り値を直接扱う部分ではこれは自明であるが、セルの一部として永続化する場合には注意が必要である。その場合は、例えば、ひとつのテーブルの内部に閉じ込めるような設計とし、それ以外の部分に紛れ込まないようにすれば、プログラミング時に赤オブジェクトの可能性があるかどうかを認識しやすい。

4.4 赤オブジェクトへの引数渡し

赤オブジェクトへのメソッド起動の引数としてオブジェクトを渡す場合は、赤オブジェクトからのそのオブジェクトのメソッドが起動できなければならないだろう。赤オブジェクトからのメソッド起動を受けられるのは赤オブジェクトだけなので、緑オブジェクトを引き渡す場合は赤オブジェクトに変更してから渡す必要がある。赤オブジェクトを保護することはできないので、破壊されると困るオブジェクトを渡す場合は、ディープコピーしてから色を変更して引き渡す必要がある。そのオブジェクトがさらに緑オブジェクトへの参照を持つ場合、この色の変更とコピーは、必要に応じて再帰的に実行しなければならない。したがって、セルを設計する際には、赤オブジェクトに対するメソッド起動の引数に、そのような再帰的な色の変更やコピーの難しいオブジェクトを与えるような設計にすべきではない。どうしてもそれが必要である場合、そのようなオブジェクトを赤オブジェクトとして引き渡すのではなく、別のセルにしてしまうことも検討するとよい。

4.5 赤オブジェクトどうしの分離

赤オブジェクトが赤オブジェクトからメソッド起動される場合は、緑オブジェクトに対するような保護機構は働かない。したがって、赤オブジェクトに別の赤オブジェクトを直接操作させると、破壊されるおそれがある。このため、セルの設計者は、セルの内部で赤オブジェクトどうしが不必要に接触しないように設計しなくてはならない。

4.6 赤オブジェクトどうしが接触する場合

前項に関わらず、セルの提供する機能の性質上、どうしてもセル内部で赤オブジェクトどうしを接触させなくてはならない場合もある [2]。この場合、セルの設計を工夫するだけでは、これらの赤オブジェクトどうしが破壊し合うのを防ぐことはできない。赤オブジェクトが破壊されることによってセルに直接の被害が生じるわけではないが、そのセルが提供するサービスが利用不能になったり品質が低下することが考えられる。これを防ぐには、このようなセルの内部で活動させる目的で送り込まれるオブジェクトを、メソッド起動によって破壊されないようにプログラムさせる。そのようなオブジェクトどうしは、他のセルで出会っても、互いを破壊することはできない。セルの設計者は、しばしばこのようなオブジェクトの抽象クラスを提供するが、以上から、その抽象クラスのインタフェースには、状態を変更するようなメソッドは含められないことが分かる。このようなオブジェクトは一般的な用途に用いるのは不便なことが多く、通常はこの目的のために特に設計されるクラスとなるだろう。

5 関連研究

OZ++[7] は、OZ に先行して開発されたオブジェクト指向分散環境であり、OZ のセキュリティモデルの原型である [8]。ただし、OZ++ ではネイティブコードが動作していたので、ホストシステムへのアクセスを適切に制限するのが困難であった。同じ問題は Active-X を用いた Java プログラムにも見られる。OZ の実装言語として Java を採用した理由のひとつでもある。

外来のプログラムによる予測できない挙動に対抗する技術としては、Java の Applet に実績がある。Applet は、ある一定の範囲の外側に対するアクセスは許可された範囲でしかできないという、sandbox モデル [9] を採用している。同じ VM 上の他のオブジェクトは、接触させないことによって保護している。クラスファイルにディジタル署名することで認証が可能だが、特定のユーザに対してひとたび権限を与えると、アクセスの種類によらず何でもできてしまい、柔軟性に欠ける。この点を解決するものであるらしい ACL API が提供されているが、JDK では使用されておらず、具体的にどのようなモデルで保護するか、および、オブジェクトの保護に利用できるのかど

うかは不明である。同様の対策としては、Netscape から、capabilities API が提案されている。これは、Applet に対して、"ファイル読み込み"、"スレッドアクセス" など、より細かいアクセス制御を可能としている。許可の単位はスタックフレームであり、スタックの上位に capability を有するフレームがあれば、アクセスが許可される。許可を申請したメソッドが復帰すると、その許可は無効になる。ホストシステムに対するアクセス制御としては柔軟になったが、やはり Applet 用なので、オブジェクトの保護については考えられていない。

sandbox は、一定以下の大きさのアプリケーションを丸ごとロードする場合には有効だが、OZ のようにネットワーク上のサーバを使って計算を進めるタイプのプログラムを記述するには向かない。分散システムを記述するには、Java 上の RMI や HORB を利用する方がよい。しかしこれらは、RMI 機能は提供するが、未知のクラスのオブジェクトを受信して動作させる仕組みは提供しない。そのような仕組みを作ることはできるが、RMI によって輸入されたオブジェクトを識別する仕組みがないので、輸入したオブジェクトが他のオブジェクトを破壊するのを防護しにくい。

6 まとめ

ネットワーク上のサービスを、相互運用性を保ったままで独立に拡張 / 変更するためには、未知のクラスのオブジェクトを受信して動作させる機能が必要だが、そのためにセキュリティ上の問題が生じるおそれがある。

OZ では、RMI の際のユーザ認証に基づくアクセス制御と、RMI によって輸入したオブジェクトを識別する機能を提供することでこの問題を解決しようとしている。このためには、セルのプログラマに適切なセキュリティ設計が求められる。本稿では、その際に役立つ指針を述べた。

OZ システムは、1997 年度末に完成する予定で開発が進められている。現在 1.0-α 版がインターネット上で公開されている [10]。

この研究は情報処理振興事業協会 (IPA) が実施している「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。

参考文献

[1] 保田, 藤野, 西岡, 塚本: "オブジェクト指向分散環境 OZ におけるフェデレーション管理の概念設計", 情報処理学会 第 53 回 全国大会, 1K-10, pp. 303–304, Mar. 1997.

[2] 西岡, 塚本: "オブジェクト交換を利用した分散サービス利用のためのフレームワーク", 情報処理学会 研究報告 "システムソフトウェアとオペレーティングシステム", Vol. 96, SWoPP 秋田 '96, Aug. 1996.

[3] Nishioka T., Hamazaki Y. and Tsukamoto M.: "The Local Class Name Space Facility for Worldwide Object-oriented Distributed Systems Environment," Worldwide Computing and Its Applications '97, Mar. 1997.

[4] 濱崎, 西岡, 塚本: "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 53 回 全国大会, 1K-7, pp. 297–298, Mar. 1997.

[5] Needham, R.M. and Schroeder, M.D.: "The Cambridge Distributed Computing System," Wokingham, England, Addison-Wesley, 1982.

[6] 濱崎, 樋口, 西岡, 塚本: "オブジェクト指向分散環境 OZ の暗号化通信プロトコル", 情報処理学会研究報告 (OS), SWoPP 阿蘇 '97, Aug. 1997.

[7] 塚本, 濱崎, 音川, 西岡: "クラスの共有と配送にもとづくオブジェクト指向分散システムの設計と実現", 情報処理学会 論文誌, May 1996.

[8] 大西, 濱崎, 西岡, 塚本: "オブジェクト指向分散環境 OZ++ におけるインターネットセキュリティ", 情報処理学会 研究報告 "マルチメディア通信と分散処理", pp. 219-226, Oct. 1995.

[9] Java のセキュリティホームページ: `http://www.javasoft.com/security/`

[10] OZ プロジェクトのホームページ: `http://www.etl.go.jp/etl/bunsan/OZ_Proj/`