コンピュータセキュリティ　10−35
(2000. 7. 25)

Java RMI を用いたファイアウォール環境下での双方向オブジェクト通信方式

寺田 雅之　　中嶋 良彰　　井口 誠
{te, yoshiaki, iguchi}@isl.ntt.co.jp

NTT 情報流通プラットフォーム研究所
〒 239−0847 神奈川県 横須賀市 光の丘 1−1

あらまし　オブジェクト指向設計/プログラミングでは，オブジェクト間の相互呼び出しが多用される．分散オブジェクト環境でこれを実現するためには，各オブジェクトが存在するホスト間で自由に通信が可能であることが求められる．しかしながら，実際のネットワーク環境では，ファイアウォールや NAT の介在などにより，通信の起点が片側に限定されるなど双方向の接続が阻害されることが多い．本稿では，Java RMI のカスタムソケット機能を応用し，オブジェクトの双方向呼び出しをアプリケーションからは透過的に TCP/IP 上の片方向接続に変換することにより，上記のような環境で双方向のオブジェクト呼び出しを可能にする方式を提案する．

キーワード　ファイアウォール，分散オブジェクト，Java, RMI

Two-way remote object communication method over firewall

Masayuki TERADA　　Yoshiaki NAKAJIMA　　Makoto IGUCHI
{te, yoshiaki, iguchi}@isl.ntt.co.jp

NTT Information Sharing Platform Laboratories
1−1 Hikari-no-oka, Yokosuka, Kanagawa, 239−0847 Japan

Abstract　In object-oriented programming, mutual referencing among objects plays a crucial role. In order to implement mutual referencing in distributed object environments, the communication path among hosts, on which the remote objects to be mutually referenced reside, should be established unconditionally. Unfortunately, such an assumption is not practical in real network environments such as the Internet, because in many cases firewalls and IP network address translators (NAT) restrict the TCP/IP connections that can be established to "one-way".

This paper presents a solution to this problem by using the Java RMI custom socket feature. The method we present solves the problem by seamlessly translating a two-way remote object invocation into one-way TCP/IP connection establishment. Once translated, the remote objects can successfully communicate over the firewall.

key words　Firewall, distributed object, Java, RMI

1. はじめに

オブジェクト指向を用いた設計やプログラミングにおいては，複数のオブジェクトが互いに呼びあうような，オブジェクト間の相互呼び出しが多用される．

有名な例としては，Smalltalk における GUI のアーキテクチャとして用いられている Model, View, Controller (MVC) の三つ組 [1] が挙げられる．

たとえば MVC では，Model が保持している値が変更されたときに Model から View に更新を通知し (Model → View の呼び出し)，View はその値を読みだすために Model にアクセスする (View → Model の呼び出し)，というように Model と View との間で相互呼び出しが用いられている．

また，このような相互呼び出しは，GUI に限らず，Observer パターンなど，オブジェクト指向におけるデザインパターンの中にも多く見ることができる [2]．

一方，近年では Java RMI や CORBA の普及により，分散オブジェクト技術が広く利用されるようになってきている．しかしながら，分散オブジェクト環境ではネットワークの制約から相互呼び出しの適用が困難となることが多い．これは，実際のネットワーク環境では，相互呼び出しを行なうオブジェクト間にファイアウォールが介在するなどの理由により，オブジェクト間での自由な双方向通信が阻害されることが多いためである．

本稿では，上記のようなネットワーク環境においても分散オブジェクト間の双方向呼び出しを可能とする方式を提案する．本方式は，ORB のトランスポート層を置換することによりこれを実現するため，既存の ORB や 既存のアプリケーションへの影響を最小限に留められるという特長を持つ．

2. 分散オブジェクトの双方向呼び出し

本章では，分散オブジェクト環境におけるオブジェクト間の相互呼び出しの有用性について述べ，ファイアウォールなどによる通信の制約からその適用が困難になることを示す．

分散オブジェクト間の相互呼び出しが有用な例として，遠隔ホスト間のイベント通知が挙げられる．

たとえば，遠隔ホストで行なわれる処理の進捗状況を，手元の画面にプログレスバー☆として表示する機構を，Java Swing[6] などの MVC モデルを採用したクラスライブラリを用いて実現しようとすると，Model である遠隔ホスト上の処理主体と View となるプログレスバーとの間で相互に遠隔呼び出しが発生することになる．

また，その他の例として，電子現金や電子チケットの流通プロトコルなどの複雑な情報のやりとりが必要なプロトコルの実現が挙げられる．

これらのプロトコルを，分散オブジェクト間の相互呼び出しを用いて実現すると，(1) 送受信するデータのエンコード/デコード処理を行なう必要がなくなる，(2) データの追加などのプロトコルの変更への対処が容易となる，(3) プロトコルの状態遷移をスレッドの実行状態として隠蔽できる，(4) 異常発生時の状態遷移を例外処理として統一的に扱える，などの利点が得られるため，その構築が容易となる．

しかしながら，近年ではセキュリティ上の理由からファイアウォールを介してインターネットに接続していたり，個人用 ISDN ルータの普及により ISP への接続に NAT (Native Address Translator) を用いていたりするようなネットワーク環境が多い [3]．それらの環境では，ファイアウォールや NAT ルータの外部から内部への接続が全く行なえないか，外部からは HTTP や SMTP など限られたサービスへの接続しか許されない☆☆ため，Java RMI など多くの ORB では内部から外部への片方向のオブジェクト呼び出ししか行なうことができない．

次章において，外部からの接続を許さないネットワーク環境でも双方向呼び出しを可能とする機能を既存の ORB に付加することにより，ファイアウォールやアプリケーションへの影響を及ぼすことなしに，ファイアウォール環境下での分散オブジェクト技術の利用を可能にする方式を提案する．

3. 実現方式

本章では，ファイアウォールなどが介在することにより外部からの接続が許されない環境において，外部からの遠隔オブジェクト呼び出しが制限される理由について示すとともに，ファイアウォール内からの TCP 接続とソケットの中継による仮想コネクションを用いることにより，そのような環境において外部からのオブジェクト呼び出しを可能とする方式を提案する．

☆ 時間がかかる処理を行なう際に用いられる，進捗状態を表示するウィジェット (GUI 部品)．

☆☆ ファイアウォールや NAT ルータが IPsec などによる VPN (Virtual Private Network) 機能を備えていれば上記が可能となるケースもあるが，それらの機器は高価であり，一般的に普及しているとは言い難い．

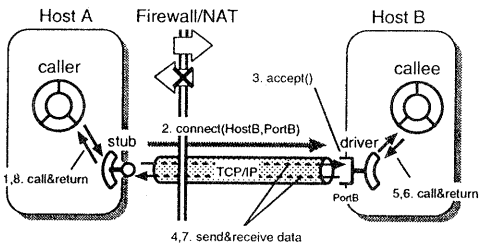

図 1　ファイアウォール内からの遠隔オブジェクト呼び出し

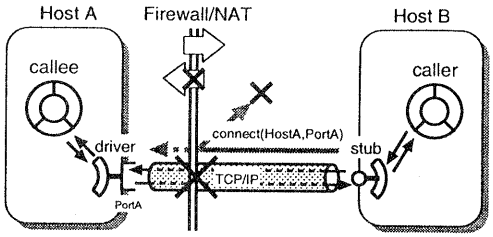

図 2　ファイアウォール内への遠隔オブジェクト呼び出し

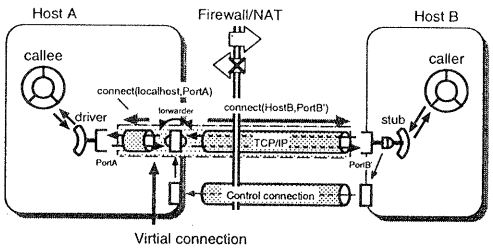

図 3　仮想コネクションを用いた遠隔オブジェクト呼び出し

3.1　ORB による遠隔オブジェクト呼び出し

分散オブジェクト環境における遠隔オブジェクト呼び出しは，ORB (Object Request Broker) と呼ばれる機構によって実現される．代表的な ORB としては，OMG による CORBA[4] や，Sun による Java RMI[5] などが挙げられる．

以下にこれらの ORB を用いた遠隔オブジェクト呼び出しの手順の例を示す (図 1)．この例では，呼び出し元 (caller) と呼び出し先 (callee) との間には caller 側から callee 側への接続のみを許すファイアウォールが介在するが，呼び出しは正常に終了する．

(1)　Caller による stub の呼び出し．
(2)　Stub から driver への TCP 接続確立 (connect(HostB, PortB)/accept())☆．
(3)　呼び出し情報 (オブジェクト識別子，メソッド名，引数) の送信．
(4)　Driver による callee の呼び出し．
(5)　Callee から driver への返値の返却．
(6)　返値の送信．
(7)　Stub から caller への返値の返却．

次に，コールバックなどにより呼び出し方向が逆転し，ファイアウォールの外から内への遠隔呼び出しが発生する場合を考える．この場合，ファイアウォールにより HostB から HostA への TCP 接続が拒絶されるため，この遠隔呼び出しは失敗する (図 2)．

したがって，このようなファイアウォールが介在する環境で双方向のオブジェクト呼び出しを可能にするためには，ファイアウォールの外部から内部への TCP 接続を用いずに遠隔オブジェクト呼び出しを実現する方式が必要となる．

3.2　仮想コネクションによる遠隔呼び出し

前節におけるファイアウォール外からのオブジェクト呼び出しの例において，呼び出しに伴う TCP 接続が拒絶されることが呼び出し失敗の原因となった．本方式では，オブジェクト呼び出しの際に，逆方向の TCP 接続を用いた仮想的な接続を確立する機能を実現することにより，そのようなファイアウォール外からの遠隔オブジェクト呼び出しを可能にする手段を提供する．

本方式における仮想コネクション確立の処理は以下のように行なわれる (図 3)．ここで，あらかじめ callee 側から caller 側に別の TCP 接続 (制御コネクション) が確立されているものとする．この制御コネクションの確立は，上位アプリケーションの初期化処理の一環として行なう．

(1)　Stub から driver への仮想コネクション接続要求 (pseudo-connect(HostA, PortA))．
(2)　Stub 上に待ち受け部を生成 (listen(PortB'))．
(3)　Stub から制御コネクション経由による中継部 (forwarder) への接続依頼の送信．
(4)　中継部から待ち受け部への TCP 接続 (connect(HostB, portB'))．
(5)　待ち受け部による接続の受理 (accept())．
(6)　中継部から driver への TCP 接続 (connect(localhost, portA))．
(7)　Driver による接続の受理 (accept())．以後，中継部は両 TCP 接続のデータを相互に中継する．

上記のようにして確立された仮想コネクションは，ファイアウォールの外側から内側への TCP 接続を伴

☆ ただし，Java RMI などの実用 ORB は，性能上の理由からある条件下でコネクションの再利用を行なうこともある．この場合，2 回目以降の呼び出しでは TCP 接続確立の処理が省略され得る．

わないため，ファイアウォールにより阻害されることはない．

この仮想コネクションを用いた遠隔オブジェクトの呼び出しは，以下のような手順で実行される．

(1) Caller による stub の呼び出し．
(2) Stub から driver への仮想コネクション確立 (pseudo-connect(HostA, PortA)/accept())．
(3) 呼び出し情報 (オブジェクト識別子，メソッド名，引数) の送信．
(4) Driver による callee の呼び出し．
(5) Callee から driver への返値の返却．
(6) 返値の送信．
(7) Stub から caller への返値の返却．

上記の手順が示すように，(2) における仮想コネクション接続要求の内部処理を除いて，通常の遠隔オブジェクト呼び出しと変わりがない．すなわち，ORB の視点からは，stub の connect() 処理の置換以外には (accept() 処理を行なう部分も含め) ORB の実装に手を加える必要はない．また，アプリケーションの視点からは，制御コネクションの初期化処理以外には影響がない．したがって，本方式を適用した ORB を既存の ORB と全く同様に利用することが可能となる．

3.3 Java RMI を用いた実装

筆者らは，本方式を Java RMI を用いて実装した．以下では，Java RMI に本方式を適用した実装である ROOF (Remote Object Over Firewall) について述べる．

ROOF は，前節で述べた connect() の置換のために，Java RMI (1.1 以降) が備えるカスタムソケットというトランスポート層の分離機構を利用している．

カスタムソケットとは，Socket 型のオブジェクトを生成するファクトリオブジェクト (Socket Factory) をリモートオブジェクトの初期化時に与えることにより，通常の通信方式のかわりにその Socket Factory が生成するソケットオブジェクト☆を用いて通信させる仕組みである [6]．

Socket Factory には，Client Socket Factory (CSF) と Server Socket Factory (SSF) の 2 種類があり，それぞれ stub 側と driver 側のソケットを生成する．

ROOF では，これらのうち CSF の置換により，本方式を用いたオブジェクト呼び出しを実現する．この置換は，リモートオブジェクトの初期化時に CSF を指定するか，初期化時などに標準 CSF との置換を指示することによって行なわれる．特に，後者の手段を用いて置換を行なうことにより，アプリケーションや Java RMI の実装コードを変更することなく，Java RMI にファイアウォールを跨がるような相互遠隔オブジェクト呼び出し機能を付与することが可能となる．

また，ROOF は上記の CSF の他，制御コネクションの確立など本方式の実現に必要な機能を 1 つの JAR (Java ARchive) ファイル (roof.jar) として提供している．これらの機能を `ROOF.init()` というメソッドを呼び出して初期化することにより，本方式を用いたオブジェクト呼び出しが利用可能となる．ここで，どのホストに対して制御コネクションを確立するべきかなどの設定情報は，起動時のプロパティもしくは設定ファイルにより与える．

したがって，既存の Java RMI を利用するアプリケーションは，以下の手順により ROOF を用いた双方向オブジェクト通信が可能になる．

(1) roof.jar を classpath に含める．
(2) `ROOF.init()` と，アプリケーションの `main()` を呼び出すような wrapper を作成する☆☆．
(3) 制御コネクションに関する設定情報をプロパティもしくは設定ファイルにより指定し，wrapper を実行する．

4. 考　　察

本章では，本方式の適用領域と安全性について考察を加える．

4.1 適用可能な環境

前章で示したように，本方式は以下の条件を満たすシステムであれば，適用可能である．

アプリケーションに対する要件 あらかじめ，クライアント側のアプリケーションが制御コネクションの確立などの初期化処理を行なうことができる．

ORB に対する要件 stub による connect() を置き換える手段が提供されている．

環境に対する要件 どちらか一方のホストからもう一方のホストに対し，直接 TCP 接続を行なうことができる．

ここで，アプリケーションに対する要件については，前章の ROOF の起動例のように，起動時に wrapper を用いることなどにより容易に充足可能である．

ORB に対する要件については，Java RMI はカスタムソケット機構によりこれを満たすことが ROOF

☆ 正確には Socket クラスを継承したクラスのオブジェクト

☆☆ ただし，現在の版の ROOF ではさらに defaultSocketFactory の置換の指示と，制御コネクション確立の指示を，陽に wrapper 内で行なう必要がある．

の実装を介して確認された．CORBA についても，message-level interceptor☆ と呼ばれる機構を用いることにより充足可能であると考えられる．また，その他の ORB についても，同様の機構が備えられていることが多い．

しかしながら，環境に対する要件については，充足できない場合も少なくない．たとえば，ファイアウォールの設定により外部には HTTP proxy などのアプリケーションプロキシを経由した接続しか許されない環境は珍しくない．

ただし，上記のような環境に対しては，httptunnel[7] などのアプリケーション層のプロトコルを用いた TCP 層のトンネリングツールを利用することにより，本方式の適用が可能になると考えられる．この実証については今後の課題である．

4.2 安 全 性

本方式は，ファイアウォール内の分散オブジェクトが提供するサービスをファイアウォールの外部にも提供することを可能にする．したがって，悪意の第三者が本方式を悪用することによって内部に侵入することを排除しなくてはならない☆☆．

本方式は，制御コネクションを接続した相手以外には呼び出しを許さないため，この接続相手へのなりすましを防止できれば，第三者による攻撃を排除できる．本方式における第三者のなりすましの機会としては，以下の 3 箇所が考えられる．

- 制御コネクションの接続時
- 仮想コネクションの確立依頼時
- データの送受信時

本方式では，上記のなりすましの防止を，以下のような構成を採ることによって実現する．

まず，制御コネクション接続時の認証によって，第一のなりすましを防止する．また，認証時に鍵交換を行ない，制御コネクションをその交換した鍵を用いて暗号化することにより第二のなりすましを，仮想コネクションを同様にその鍵で暗号化することにより第三のなりすましを，それぞれ防止できる．

ROOF では現在，これらの認証や暗号化を RSA や FEAL を用いた独自の方式により行なっているが，適当な TLS (Transport Layer Security) ライブラリを利用することによって，一般的な X.509 形式の CA 証明書を用いた認証を用いることも容易である．

5. 他手法との比較

本稿では，ファイアウォールが外部からの接続を許さない環境において分散オブジェクトの相互呼び出しを可能にするための方式を提案した．しかしながら，ファイアウォールの設定変更やアプリケーションの設計変更などにより，本方式を用いずにシステムを構築することも可能である．本章では，これらの手法による対処について議論する．

なお，以下の説明において，ファイアウォール内のホストをクライアント，ファイアウォール外のホストをサーバとする．

5.1 ファイアウォールでの対応

もしファイアウォール管理者の協力が得られるなら，以下のような対処をファイアウォールに行なうことにより，外部からのオブジェクト呼び出しが可能になる．

直接通信の許可 ファイアウォール外からクライアントへの直接通信を許すようにフィルタの設定を変更する．ただし，クライアントは global IP アドレスが割り当てられている必要がある．

Proxy の設置 ファイアウォール外からのオブジェクト呼び出しを中継するような，アプリケーションゲートウェイをファイアウォール上に設置する．ただし，呼び出し元であるサーバに proxy の存在を意識させるための手段が必要になる．

固定 NAT の利用 NAT によるアドレス変換が通信の阻害要因となっている場合，固定 NAT 機能を用いてクライアントのポートを外部接続側のポートにマッピングする．ただし，複数クライアントによる利用は困難となる．

しかしながら，上記の対処はいずれもファイアウォール自体に恒常的な穴開けを行なうことになる．そのため，セキュリティ上の観点や管理コストの増大の面から，これらの対処についてファイアウォール管理者の協力を得ることは一般的に大変な困難を伴なう．

5.2 アプリケーションの設計変更

上記のような環境の変更による対処が困難である場合，アプリケーションプログラムに対して以下のような設計変更を行なうことにより，双方向呼び出しを排除することが可能である．

フローの変更 あらかじめ，サーバからクライアントを呼び出すタイミングがクライアントにとって明らかな場合，アーキテクチャやフローを変更することによってサーバからの呼び出しをクライアン

☆ 文献 [4], Chap.21.

☆☆ 実際には，さらに接続相手による権限外のアクセス (root 権限の奪取など) についても考慮する必要があるが，これは buffer overrun の防止や不用意な root 権限での実行の排除，もしくは Java の砂箱 (SandBox) モデルの利用など，アプリケーションの実装方式自体の課題であるため，本稿では割愛する．

トからの呼び出しに変更することができる．しかしながら，プログレスバーの例のようにサーバからクライアントを呼び出す機会が不明確な場合にはこの対処は適用できない．また適用できる場合であっても，不自然な呼び出し関係の変更によりクラス構造が不均整になりがちなため，モジュールの再利用性が損なわれ，規模の増大を招くとともに以後の修正が困難となる．

ポーリングの利用　クライアントからサーバに対して，定期的に処理要求の有無を確認するポーリングを行ない，要求が有った場合に処理を行なうような機構を用意することにより，任意の時点でのサーバからの呼び出しを擬似的に実現できる．しかしながら，ポーリングによる無駄な通信の発生，ポーリング待ちによる処理遅延などの問題がある．

下位プロトコルの直接利用　分散オブジェクト技術の利用をあきらめ，TCP などの下位プロトコルを直接利用する選択肢も考えられる．しかしながら，第2章で述べたような分散オブジェクト利用時の恩恵が受けられなくなるため，アプリケーションの設計/実装コストが高価になりがちである．

このように，アプリケーションプログラムの設計による対処では，設計コストや再利用性，および実行時間などの点で問題が生じることになる．

6. まとめ

本稿では，分散オブジェクト間の双方向呼び出しの必要性と，ファイアウォールや NAT がこれを阻害する原因となることについて示した．この問題に対し，呼び出しとは逆方向からの TCP 接続を用いて仮想コネクションを構築することにより，ファイアウォール環境下でも双方向のオブジェクト呼び出しを可能にする方式を提案した．また，本方式を Java RMI のカスタムソケット機構を用いて実装することにより，Java RMI に本方式が適用可能であることを実証した．

筆者らは，電子権利流通システム FlexTicket[8] を本方式の実装である ROOF を利用して構築することにより，本方式の実用性を確認した．FlexTicket は，XML を用いて階層的に記述された権利定義を，原本性を保証しつつ移送する[9] という比較的複雑なプロトコルを有する．しかしながら，ROOF を用いることによりファイアウォールを意識することなく分散オブジェクト技術を最大限に利用した設計が可能となったため，その実装はモジュール性が高く，簡素な構成とすることができた．

ただし，現状の ROOF は仮想コネクションの確立に TCP 接続を必要とするため，ファイアウォールの外部に対して接続可能なプロトコルが HTTP などに限定されている環境では利用できないという問題がある．また，仮想コネクションの確立に複雑な手順を要するため，低速な回線上では仮想コネクションの確立のためのオーバヘッドが無視できないと予想される．

そのため，HTTP を用いた仮想コネクションの確立など TCP の直接通信が行なえない環境への適用や，性能向上のための投機的な仮想コネクション確立の実装を現在進めており，これらの実現と評価が今後の課題である．

謝辞

本研究は，権利流通基盤 FlexTicket の研究の一環として行なわれた．本方式の実現に協力頂いた庄田隆司氏に感謝する．

参考文献

1) Krasner, G. E. and Pope, S. T.: A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80, *Journal of Object Oriented Programming*, Vol. 1, No. 3, pp. 26–49 (1988).
2) Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns*, Addison-Wesley Publishing (1995). (監訳: 本位田真一，吉田和樹 訳: “デザインパターン”，ソフトバンク (1995)).
3) Cheswick, W. R. and Bellovin, S. M.: *Firewall and Internet Security: Repelling the Wily Hacker*, AT&T Bell Laboratories (1994). (翻訳: 田和勝，鎌形久美子 監訳: 川副博 訳: “ファイアウォール”，ソフトバンク (1995)).
4) Object Management Group: *The Common Object Request Broker: Architecture and Specification, Revision 2.3.1* (1999).
5) Sun Microsystems: *Java Remote Method Invocation Specification*, revision 1.7, java2 sdk, standard edition (1999).
6) Sun Microsystems: *Java2 Platform, Standard Edition, v1.3 API Specification* (1999). http://java.sun.com/j2se/1.3/docs/api/index.html.
7) Brinkhoff, L.: *GNU httptunnel 3.0.2* (2000). http://www.nocrew.org/software/httptunnel.html.
8) 水野康尚，千綿伸之，大嶋嘉人，松山一雄: 権利流通基盤実現のための汎用デジタルチケットシステム，コンピュータセキュリティシンポジウム'99 予稿集 (1999).
9) 寺田雅之，久野浩，花舘蔵之: 権利流通基盤実現のための原本性保証方式，コンピュータセキュリティシンポジウム'99 予稿集 (1999).

盛光印刷所