ファイアウォール環境のための双方向オブジェクト通信機構 ROOF の実装と評価

寺田雅之[†] 中嶋良彰[†] 井口誠[†]

ファイアウォールや NAT が介在することが多い近年のインターネット環境では，Java RMI などの分散オブジェクト機構の利用は困難である．これは，ファイアウォールなどが外部から内部へのTCP 接続を制限することが多いため，オブジェクト指向で多用される双方向のオブジェクト呼び出しが行なえるとは限らないことによる．筆者らは分散オブジェクト機構のトランスポート層を置換し，呼び出し方向とは逆方向からの TCP 接続を用いた仮想コネクションを用いてオブジェクト呼び出しを行なうことによりこれを解決する方式を提案している．本稿では，Java RMI のカスタムソケット機能を利用した上記方式に基づく双方向オブジェクト通信機構 ROOF の構成と，その性能評価の結果について述べる．

Implementation and Evaluation of ROOF: A System for Two-way Remote Object Communication over Firewall

Masayuki Terada,[†] Yoshiaki Nakajima[†] and Makoto Iguchi[†]

Firewalls and NATs are being employed increasingly in the Internet. This trend has made the application of the distributed object mechanism difficult; the incoming TCP/IP connections from the outside of the firewalls are often prohibited, and the restrictions make the realization of two-way remote object communication complicated. We have proposed a mechanism to solve the problem by designing a new transport layer, which seamlessly translates two-way remote object invocation into one-way TCP/IP connection establishment.

In this paper, the implementation of ROOF, a system that realizes our mechanism using the Java custom socket feature, is discussed along with its performance evaluation.

1. はじめに

分散オブジェクト技術は，複雑な通信のやりとりを行なう高度なネットワークサービスの迅速な構築に有用である．

しかしながら，分散オブジェクトを利用して構築したサービスをインターネット上で提供することは困難であることが多い．その理由として，オブジェクト指向プログラミングにおいてはオブジェクト間の相互呼び出しが多用される[1]にもかかわらず，インターネット環境では双方向の遠隔オブジェクト呼び出しが行なえるとは限らないことが挙げられる．

たとえば，セキュリティ上の理由からファイアウォールが設置されている環境や，ISP への接続に NAT を用いている環境では，ファイアウォールや NAT ルータの外部から内部への接続は全く行なえないことが多い[2]．Java RMI などの遠隔オブジェクト呼び出しに TCP 接続を伴なう分散オブジェクト機構 (ORB) を用いた場合，このような環境ではファイアウォールの外部から内部へのオブジェクト呼び出しは不可能となる．

そこで筆者らは，呼び出しの方向とは逆方向の TCP 接続を用いた仮想コネクションを確立するトランスポート層を構築し，これを既存の ORB のトランスポート層と置換することにより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した[3]．上記方式は，既存の ORB に対する機能付加によりこれを実現するため，適用に際して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．

筆者らは，Java RMI を用いて上記の方式に基づく双方向オブジェクト通信機構 ROOF (Remote Object Over Firewall) の実装を行なった．本稿では，ROOF の設計方針と実装方式について述べるとともに，性能について評価を行なう．

† NTT 情報流通プラットフォーム研究所
NTT Information Sharing Platform Laboratories

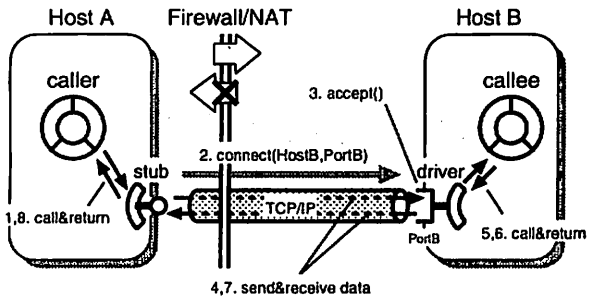

図 1　ファイアウォール内からの遠隔オブジェクト呼び出し

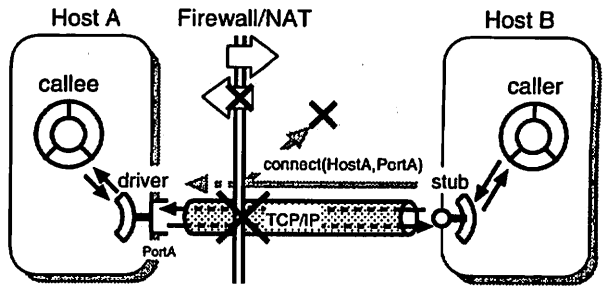

図 2　ファイアウォール内への遠隔オブジェクト呼び出し

2. Java RMI の遠隔オブジェクト呼び出し

Java RMI[4] は，Java SDK の一部として配布されている分散オブジェクト機構であり，その API は J2SE (Java2 Platform, Standard Edition) の標準 API に含まれる [5]．Java 言語の急速な普及に伴ない，Java RMI を用いた分散オブジェクト環境は広く利用されつつある．

Java RMI を用いた遠隔オブジェクト呼び出しは，図 1 に示す手順で行なわれる☆．ただし，Java RMI はコネクションの再利用機構を備えており，一定時間以内に連続した遠隔オブジェクト呼び出しを行なう場合には，以前に確立されたコネクションを再利用して通信を行なう．同図が示すように，呼び出し元 (caller) と呼び出し先 (callee) との間にファイアウォールが介在しても，呼び出し方向と同方向の TCP 接続が許可されていれば呼び出しは正常に実行できる．

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部から内部への遠隔オブジェクト呼び出しが発生したとする．この場合は，ファイアウォールにより呼び出しに伴う 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，上記のようなファイアウォールが介在するホスト間では，Java RMI を用いた双方向の遠隔オブジェクト呼び出しを行なうことができない．

3. 実 装 方 式

前章で述べたように，通常の Java RMI による遠隔オブジェクト呼び出しは，呼び出し元から呼び出し先への TCP 接続を伴なうため，外部からの TCP 接続が遮断される環境内のオブジェクトに対して，外部から遠隔オブジェクト呼び出しを行なうことはできない．

ROOF は上記のような環境において双方向の遠隔オブジェクト呼び出しを可能とするために，Java RMI が備えるトランスポート層の置換機能 (カスタムソケット) を用いて，呼び出し方向とは逆方向の TCP 接続を含む仮想コネクションを介して遠隔オブジェクト呼び出しを行なう機能を提供する．図 3 に，ROOF における遠隔オブジェクト呼び出し機構の概略を示す．

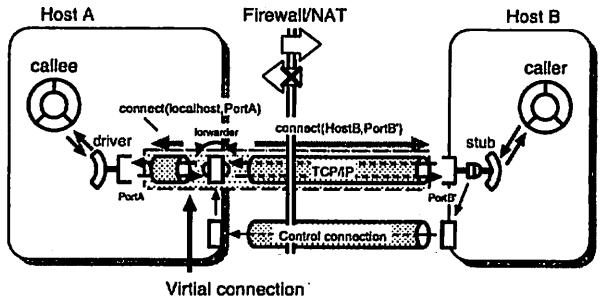

図 3　ROOF における遠隔オブジェクト呼び出し

ROOF を用いてファイアウォール外のホスト (HostB) からファイアウォール内のホスト (HostA) を呼び出すためには，あらかじめ HostA から HostB に対して TCP 接続を行なう．この TCP 接続を制御コネクションと呼ぶ．制御コネクションの確立は，HostA におけるアプリケーションの初期化時などに一度だけ行なえば良い．

HostA から HostB に制御コネクションが確立されると，HostB のオブジェクト (caller) は HostA 上のオブジェクト (callee) への遠隔呼び出しの際に，通常の TCP 接続を行なうかわりに仮想コネクションを確立する．この仮想コネクションの確立は，HostB から制御コネクションを介して HostA の中継部 (forwarder) に対して仮想コネクションの確立要求を行ない，forwarder から caller と callee の双方に TCP 接続を行なうことにより行なわれる．

以下では，制御コネクションの確立機構，仮想コネクションの確立機構，仮想コネクションを介した遠隔オブジェクト呼び出し機構のそれぞれについて，その構成と実行手順を説明する．

☆ 図中の driver は，Java RMI では skelton という言葉を用いて表現されている．

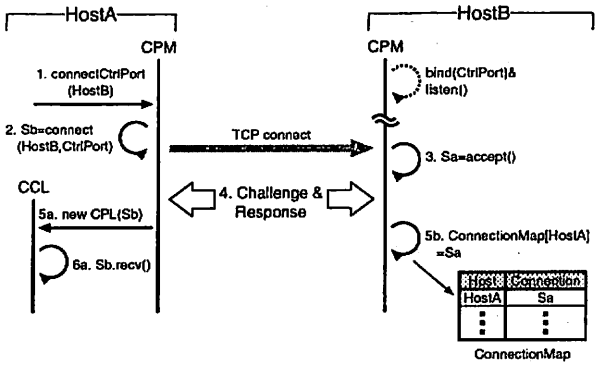

図 4　制御コネクションの確立手順

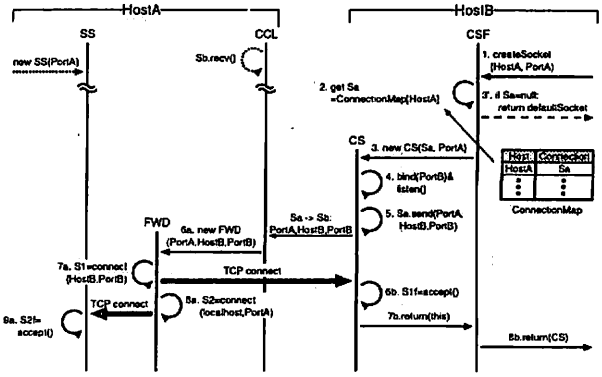

図 5　仮想コネクションの確立手順

3.1　制御コネクションの確立手順

図 4 に，制御コネクションの確立手順を示す．

HostA から HostB への制御コネクションの確立は，HostA の CPM (Control Port Manager) に対して HostB の名前を指定することによって行なわれる (手順 1)．HostA の CPM は HostB の CPM に対して TCP 接続を確立し (手順 2-3)，RSA 暗号を用いてホスト認証と制御/仮想コネクション暗号化用の FEAL 鍵の共有を行なう (手順 4)☆．上記の手順により確立された TCP 接続が，制御コネクションとなる．

制御コネクションの確立に成功すると，HostA は CCL (Control Connection Listener) に制御コネクションの端点となるソケット Sb を監視させる (手順 5a-6a)．また，HostB 側のソケット Sa は HostA をキーとしてテーブル ConnectionMap に保存される (手順 5b)．

3.2　仮想コネクションの確立手順

図 5 に，前節に示した手順により確立された制御コネクションを用いて，HostB から HostA への仮想コネクションを確立する手順を示す．仮想コネクションの確立は，遠隔オブジェクト呼び出しの際に HostB 上の stub により必要に応じて行なわれる．

まず，HostB の stub から CSF (Client Socket Factory) に対して ⟨HostA, PortA⟩ への接続確立が指示される (手順 1)．ここで，PortA はこの stub に対応する HostA の driver が待ち受けをしているポートである．HostA から制御コネクションが確立されていない場合，CSF は通常のソケットを返却する (手順 2-3')．この場合，HostA に対しては通常の RMI 呼び出しが行なわれることになる．

HostA からすでに制御コネクションが確立されていた場合は，CSF は HostA に対する制御コネクションの端点 Sa と PortA を引数として，CS (Client Socket) を生成する (手順 3)．CS は PortB に bind されたソケットを生成し，制御コネクションに ⟨PortA, HostB, PortB⟩ の組を送信するとともに PortB の待ち受けを行なう (手順 4-5)．PortB は，ROOF が定める任意のポートである．

制御コネクションからの要求を受理した CCL は，受理した要求 ⟨PortA, HostB, PortB⟩ を引数として FWD (forwarder) を生成する (手順 6a)．FWD は，CS ⟨HostB, PortB⟩ と SS (Server Socket) ⟨localhost, PortA⟩ にそれぞれ TCP 接続を行なう (手順 7a-8a)．なお，FWD–CS 間の接続は，制御コネクション確立時に交換した FEAL 鍵により暗号化を行なう．以後，FWD は 2 つのスレッドを生成し，両 TCP 接続から送られてくるデータを相互に中継する．

CS は FWD からの接続を受理すると CSF に (生成子の返値として) 自らを返却する (手順 6b-7b)．CSF は CS を呼び出し元の stub に返却する (手順 8b)．また，SS も FWD からの接続を受理する (手順 9a)．これにより，CS–SS 間に全二重通信が可能な仮想的な接続が確立される．

3.3　遠隔オブジェクト呼び出し手順

図 6 に，前節に示した手順により確立された仮想コネクションを用いて HostB のオブジェクトから HostA のオブジェクトの遠隔呼び出しを行なう手順を示す．

Stub に対して遠隔オブジェクト呼び出し要求がなされると，stub は引数オブジェクトなどの直列化やヘッダの付与などを行ない，CS に対してそれらの書き込みを行なう (手順 1)．CS は書き込みが行なわれたデータを FWD に送信し，FWD は SS にそのデータを中継する (手順 2-3)．Driver は SS からデータを読み込むと，直列化されたデータを復元して呼び出し

☆ ただし，本稿ではこれらの鍵の管理機構や利用手順などについては割愛する．

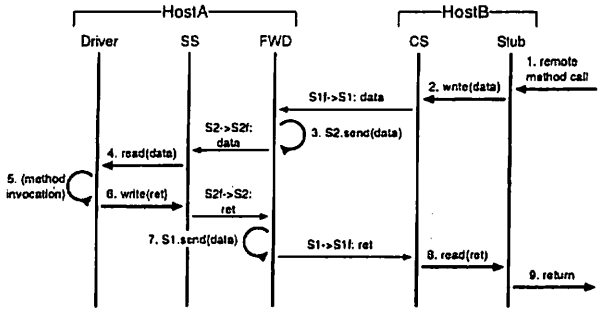

図 6　遠隔呼び出し手順

表 1　制御コネクション確立

No.	計測対象	平均 (ms)	標準偏差 (ms)
C-1	ROOF	503	44
C-2	ROOF(認証なし)	32	29

対象メソッドを実行し，返値や例外を直列化して SS に書き込みを行なう (手順 4-6).

SS から stub までの返値の送信は，呼び出しの際と逆の手順で行なわれる (手順 7-9).

4. 性能評価

本章では，(1) 制御コネクションの確立処理，(2) 仮想コネクションの確立処理，(3) 遠隔オブジェクト呼び出し処理のそれぞれについて処理時間の計測を行なった結果について示し，その評価と考察を行なう.

本計測は，以下に示す環境で行なった.

HostA　PC/AT 互換, Pentium 200MHz☆, Windows 98, Sun JRE 1.2.2

HostB　PC/AT 互換, PentiumPro 200MHz, Windows NT4.0, Sun JRE 1.2.2

ネットワーク　10Mbps Ethernet

以下において，各処理について 21 回の連続実行を行ない，(JIT の実行やクラスの動的ロードによる遅延の影響を排除するため) 最初の 1 回を除く 20 回の計測値☆☆について平均と標準偏差を示し，その評価と考察を行なう．なお，本計測では ROOF の暗号化機能は用いていない.

4.1 制御コネクション確立処理

表 1 に，制御コネクション確立処理の計測結果を示す．本計測では，制御コネクションの確立の際に認証を行なう場合 (C-1) と，認証を行なわない場合 (C-2) のそれぞれについて計測を行なった.

認証を行なう場合では，制御コネクションの確立に 500 ms 程度を要している．認証を行なわない場合と比較すると，処理時間のほとんどが認証処理に費されていることがわかる.

ただし，本処理はアプリケーションの初期化時など，ROOF を用いた遠隔オブジェクト呼び出しを行なう前に (ホストごとに) 一度だけ実行すれば良いため，この値は実用上問題とならないことが多いと考える.

表 2　仮想コネクション確立

No.	計測対象	平均 (ms)	標準偏差 (ms)
V-1	ROOF	187	17
V-2	RMI	13	4

4.2 仮想コネクション確立処理

表 2 に，仮想コネクション確立処理の計測結果を示す．本計測では，ROOF の仮想コネクション確立処理 (V-1) と，Java RMI の標準ソケットによる接続の確立処理 (V-2) について計測を行なった.

標準ソケットによる接続が 10 ms 前後で行なわれていることと比較して，仮想コネクションの確立には 190 ms 程度を要している．しかしながら，Java RMI は接続の再利用を行なう機能を有しているため，ほとんどの呼び出しではこの影響を受けることはない．ただし，たとえば無通信状態が続いた後の呼び出しや，レジストリからリモート参照を取得する時に発生する暗黙の呼び出し☆☆☆など，接続の再利用が行なわれない呼び出しの際には，遅延が生じる要因となる.

上記の遅延は，呼び出し対象オブジェクトの参照関係がある程度固定的であり，ほとんどの遠隔呼び出しにおいて接続の再利用が行なわれる場合には問題とならないが，頻繁にリモート参照の取得を繰り返す場合には無視できない性能劣化の要因となる．後者の場合における性能劣化は，仮想コネクションをあらかじめいくつか投機的に確立してプールしておき，必要に応じてプールした仮想コネクションを払い出す機能を ROOF に追加することにより，回避できると考えられる.

ただし，後者のような構成を取るシステムの大部分は，Mediator パターンや Facade パターン [1] などの導入により参照対象を減らすことが可能であり，かつ設計上も望ましいと考える.

4.3 遠隔オブジェクト呼び出し処理

表 3 に，引数と返値がそれぞれ 10 バイトのデータから構成されるメソッドに対する，遠隔呼び出し処理

☆ ただし，測定の都合により遠隔オブジェクト呼び出し処理の測定のみ Pentium 166MHz の機器を用いている.

☆☆ ただし，遠隔オブジェクト呼び出し処理のみ 51 回の実行を行ない，50 回の計測値を用いている.

☆☆☆ レジストリからリモート参照を取得すると，分散 GC 機構から参照先のオブジェクトに対して遠隔呼び出しが行なわれる．この呼び出しには既存の接続は再利用されず，新規に生成された接続が利用される.

表 3 遠隔オブジェクト呼び出し (10 bytes)

No.	測定対象	平均 (ms)	標準偏差 (ms)
I1-1	ROOF	65	56
I1-2	RMI	60	49

表 4 遠隔オブジェクト呼び出し (10k bytes)

No.	測定対象	平均 (ms)	標準偏差 (ms)
I2-1	ROOF	268	84
I2-2	RMI	117	50
I2-3	Socket (Java)	115	93
I2-4	Socket (C)	51	5

の計測結果を示す．本計測では，ROOF の仮想コネクションを用いた呼び出し (I1-1) と，RMI の標準ソケットを用いた呼び出しの両方について計測を行なった．

どちらの呼び出しも 60 ms 程度の時間で行なわれており，ほぼ同等の遅延時間となっている．

次に，表 4 に引数と返値のデータ量をそれぞれ 10,000 バイトとした場合の計測結果を示す．本計測では比較のため，ROOF (I2-1) と RMI (I2-2) に加え，Java 上でソケットを直接使って 10,000 バイトのデータを送受信した場合 (I2-3) と，C でソケットを使って送受信した場合についても計測を行なった．

この計測では，ROOF は RMI と比較して半分程度のスループットしか得られていない．これは，音声や画像など大量のデータを送受信するようなアプリケーションを構築する際に性能劣化の要因となる．その原因としては，HostA における TCP 接続の中継処理が間に合っていないこと，もしくはソケットの実装における不適切なバッファリングなどによって中継部で処理の遅延が発生していることなどが考えられる☆．

このスループットの低下は，Server Socket に中継部の機能を統合し，driver が stub に対して直接 TCP 接続を確立する構成を採ることにより，回避可能であると考えられる．この統合を行なうためには，Client Socket の置換に加えて Server Socket の置換が必要となる．

5. 関連技術

本章では，Java 上で利用可能な主要 ORB である Java RMI や CORBA，HORB におけるファイアウォール環境への対応方式について述べる．

5.1 Java RMI のファイアウォール対応状況

Java RMI を利用する際に，呼び出し先ホストへの TCP 接続が遮断されている場合に用いることが可能な通信手段としては，HTTP による RMI プロトコルのラッピング機能の利用と，JDK 1.0.2 互換モードによるコネクション多重化の機能の転用が挙げられる．

5.1.1 HTTP によるラッピング

Java RMI は，TCP/IP による直接接続が行なえない環境で利用される場合を想定して，以下のように HTTP を用いた段階的なフォールバックを行なう機能を有している．

(1) 通常の TCP/IP による直接接続を試みる．
(2) (1) に失敗した場合，RMI プロトコルを構成するデータを HTTP POST メソッドを用いてラッピングし，リモートオブジェクトに送る．
(3) (2) に失敗した場合，標準の http ポート (port 80) に HTTP POST でラッピングしたデータを送り，それを受けた httpd から CGI 経由でリモートオブジェクトを呼び出す．

上記のフォールバックを行なうことにより，HTTP プロトコル用のアプリケーションゲートウェイ (HTTP proxy) を介した通信が可能になる．

しかしながら，上記の方式を適用するためには，ファイアウォールが外部から該当ホストへの HTTP 接続を受け付ける必要がある．また，フォールバックを用いた接続は非常に低速であるとされている [4]．

5.1.2 JDK 1.0.2 互換モードの利用

JDK 1.1 以降に含まれる RMI では，既存の JDK 1.0.2 との互換性を目的としてコネクション多重化の機能が追加されている．

本来，このコネクション多重化機能はファイアウォール対応のための手段ではないが，Sun による RMI の FAQ[6] では，この機能を転用☆☆することにより，ファイアウォールの外側から内側への呼び出しが可能になる場合があることを示唆している．

しかしながら，このモードによる接続は，タイムアウトがなく，スケイラビリティにも乏しいため，推奨されずサポートの対象外であるとされている．

5.2 CORBA IIOP

通常では，Java RMI はその標準通信プロトコルである JRMP (Java Remote Method Protocol) と呼ばれるプロトコルにより通信を行なうが，スタブのコンパイル時にオプション指定を行なうことにより CORBA IIOP (Internet Inter-ORB Protocol)[7] による通信が可能となる．これは RMI over IIOP (RMI/IIOP) と呼ばれる．ただし，RMI/IIOP では分散 GC 機能を用いることができないなど，Java RMI が備える機

☆ Java と C とのソケット性能比が 2 倍以上あることを勘案すると，後者の可能性が高いと思われる．

☆☆ 実際には，同文書には「悪用 (abuse)」と記されている．

能にいくつかの制約が生じる[8].

IIOP は，第 1.2 版以降において Bi-directional IIOP と呼ばれる (RMI における JDK 1.0.2 互換モードと同様な) 接続多重化の機能が仕様化されている．ただし，本稿執筆時点の最新版である RMI/IIOP 1.0.1 は IIOP 1.0 準拠であり，Bi-directional IIOP を利用することはできない．

上述のように，Bi-directional IIOP は本方式と異なり，接続多重化によりファイアウォールの外部から内部への接続手段を提供している．しかしながら，接続多重化による接続は，(1) ディスパッチによる性能劣化が避けられない，(2) backlog や timeout などのソケットが提供する資源管理機構を利用できなくなるため，コネクションの管理機構の実装に細心の注意を要する☆，などにより，本方式と比較して実装に留意が必要になると考える．

5.3 HORB invitation

HORB[9] は，リモート参照ごと (スタブごと) にソケットを生成するという特徴を持ち，この特徴を利用した invitation と呼ばれるサーバからクライアントへのコールバックのための機構を備えている．すなわち，HORB ではサーバ上でクライアント上のリモートオブジェクトに対するスタブを生成する際に，クライアント側から陽に指示して TCP 接続を行なうことによって，サーバからクライアントへの呼び出し (invite) を可能にする．

ただし，HORB によるソケットの利用法は Java RMI による利用法と大きく異なるため，HORB の invitation 機構をそのまま Java RMI に適用することはできない．

また，HORB において invitation 機構を利用するためには，アプリケーションの作成時に「このアプリケーションはクライアント側かサーバ側か」ということを強く意識した上で複雑な呼び出し手順を経る必要があるため，利用が容易であるとは言い難い☆☆．

6. ま と め

本稿では，外部からの接続を遮断するようなファイアウォールが介在する環境において，双方向の遠隔オブジェクト呼び出しを可能にする機構である ROOF の構成と実現方式について述べるとともに，ROOF の性能測定の結果について示し，本機構の性能への影響について議論した．

その結果から，接続の再利用が有効に働き，音声や画像など大量のデータを遠隔呼び出しの引数や返値として扱わない場合には，通常の Java RMI と比較して大きな性能劣化を生じることなく，上記のような環境における双方向の遠隔オブジェクト通信を実用的な性能で行なうことができると考えられる．

また，仮想コネクションの投機的な確立機構と，中継部と統合された Server Socket を ROOF に実装することにより，上記の条件を満たせない場合でも通常の RMI 呼び出しと同等の性能を得ることが可能であると予想される．

現在，筆者らは ROOF に対する上記の機構の実装を進めており，これらの実現と評価が今後の課題である．

☆ 文献[7] においても，実装上の留意事項としてデッドロックの可能性などが指摘されている．また，Sun が JDK 1.0.2 互換モードの利用を推奨しない理由も，このあたりの実装が不十分なためではないかと推測する．

☆☆ ただし，本稿執筆時点で HORB の invitation の利用の複雑さについては改良が進んでおり[10]，試験利用可能な β 版が公開されている．

参 考 文 献

1) 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)).
2) 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)).
3) 寺田雅之, 中嶋良彰, 井口誠: 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, 第 10 回 CSEC 研究発表会 (2000).
4) Sun Microsystems: *Java Remote Method Invocation Specification*, revision 1.7, java2 sdk, standard edition (1999).
5) Sun Microsystems: *Java2 Platform, Standard Edition, v1.3 API Specification* (1999).
6) Sun Microsystems: *Frequently Asked Questions - RMI and Object Serialization* (2000).
7) Object Management Group: *The Common Object Request Broker: Architecture and Specification, Revision 2.3.1* (1999).
8) Sun Microsystems: *RMI-IIOP Programmer's Guide* (1999).
9) HORB Project: HORB 日本語ホームページ (2000). http://horb.etl.go.jp/horb-j/.
10) 萩本順三, 不破康人: HORB 2.0 新機能の紹介, 第 3 回 HORB シンポジウム予稿集資料 (2000).