# 分散オブジェクト実行環境の軽量化

3F−3

中村 隆幸　武本 充治　田中 聡　久保田 稔

NTT 光ネットワークシステム研究所

## 1 はじめに

近い将来の交換ノードシステム、及び将来の通信ネットワークアプリケーションに適用するため、我々は分散オブジェクト指向実行環境について検討を行っている。ネットワークに適用するにあたってはオープン性が必須条件となる。このため OMG によって標準化された開かれた分散オブジェクト仕様である CORBA[1] に準拠した Object Request Broker (ORB) の、交換ノードシステムへの適用を検討している。この際従来の交換ノードシステムで求められる以下の基準を満たす必要がある。

a. 動作が軽量で高多重度処理に向くこと。
b. 信頼性・耐故障性が高いこと。
c. 制御網・管理網等複数プロトコルをサポートできること。
d. ORB 自体の保守性が高いこと。

本稿では a. に着目して、既存 ORB の分析をもとに、軽量動作する ORB を設計するための指針について述べる。さらに、その設計指針を既存 ORB の軽量化に適用し、処理性能が改善されることを示す。

## 2 背景

本節では CORBA 準拠 ORB の構成、および本稿の検討対象として用いた ORB の概略について述べる。

### 2.1 ORB の一般的な構成

CORBA 2.2 仕様に準拠する ORB の構成を示す。クライアントがサーバーオブジェクトに対してリクエストを発行すると、クライアントスタブ (stub)・ORB コアを経由し、ネットワークにメッセージが送出される。サーバー側はそれを受信し、ORB コアが処理対象となるオブジェクト及びメソッドを選択して、合致するサーバースケルトン (skeleton) を経由し、アプリケーションのメソッド実装を呼ぶ。リクエストメッセージの処理のうち、ORB コアはヘッダを、stub・skeleton は引数部分を担当する。返答メッセージは、上記と逆の経路でクライアントに戻される。

### 2.2 検討対象とする ORB

本稿では ORB の構成を検討するための土台として、我々が開発した C++ 用 ORB である DONA DPE カーネル プロトタイプ版 (以下プロトタイプ版)[2] を用いた。プロトタイプ版は、Sun 社の公開している IIOP 実装 [3] 1.1 版をもとに、市販 ORB 製品に準拠した stub, skeleton を組み合わせた構成になっている。プロトタイプであるため、Any 型等のサポートされていない機能があり、また詳細なエラーチェックも行っていない。プロトタイプ版の目的は ORB の動作検証にあるため、動作の軽量化 (チューニング) は行っていない。

## 3 ORB の処理軽量化方式

我々はプロトタイプ版について、引数の個数の少ないオペレーション処理の実行をトレースし、分析を行った。その結果、ORB コア内部でのリクエストヘッダの生成・解析に、処理時間の大半を費やしていることが判明した。さらに、不必要・非効率的な処理を洗い出し、その傾向から以下に示すような ORB 軽量化の設計指針を得た。

1. ORB コア及び stub・skeleton は、メッセージバッファ操作のために下位レイヤ関数を直接呼ぶ。
2. ループ処理中で下位レイヤの関数を繰り返し呼ぶ場合は、下位レイヤに繰り返し処理を行う専用インターフェースを設ける。
3. 動的記憶の確保 (malloc) を極力排除する。
4. 不要なオブジェクトの構築・破棄処理を避けるため、クラスの継承関係を必要最小限にとどめる。

設計指針 1 は、汎用的かつ複合的な処理を行う上位レイヤでなく、シンプルな処理を行う下位レイヤを使うという意味である。これは速度を大きく改善する反面、ORB の保守性を下げる欠点がある。上位レイヤの提供するインターフェースをシンプルにして、関数のインライン展開を利用することにより、この問題に対処する。

設計指針 2 も同様に、下位レイヤに際限なく特殊インターフェースを増やすと、ORB の保守性が低下する。そこで、文字列送受信処理 (char 送受信のループ) など、汎用的で影響の大きいものだけを軽量化の対象とする。

## 4 既存 ORB の軽量化実験

本節ではプロトタイプ版を用いて、前節の方式の検証を行う。前節で示した 4 つの設計指針をもとに、プロトタイプ版に対し軽量化を施し、その効果を検証する。

軽量化前のプロトタイプ版は、Sun IIOP 実装の上位レイヤルーチンを使用して実装しており、実行速度に対する影響が大きい。また、ORB 自体がオブジェクト指向に基づいて設計されているため、処理中にオブジェクトの動的確保が多数発生している。これらの処理を軽量化する。

An Optimization of a Distributed Object Environment
Takayuki Nakamura, Michiharu Takemoto, Satoshi Tanaka and Minoru Kubota
NTT Optical Network Systems Laboratories
3-9-11 Midori-cho, Musashino-city, Tokyo, 180-8585, Japan.
takayuki@ma.onlab.ntt.co.jp

```
void set(in short n, in short m, in long value);
long get(in short n, in short m);
void set20(in short n, in short m, in long value, in long arg1, ..., in long arg20);
void set40(in short n, in short m, in long value, in long arg1, ..., in long arg40);
void set60(in short n, in short m, in long value, in long arg1, ..., in long arg60);
```

図 1: 実験プログラムの IDL 記述

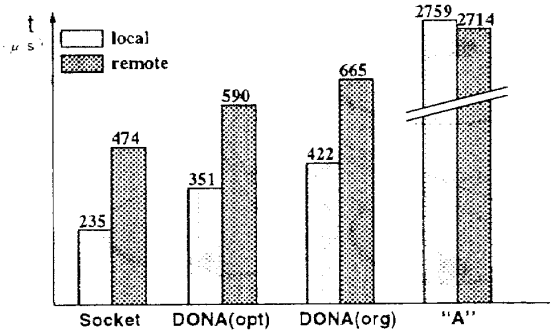


図 2: 引数の少ない場合の平均実行時間

具体例を挙げる。ORB コア内部のヘッダ生成・解析ルーチン中で、受信したヘッダ中の構造体列を解析する上位レイヤルーチン CDR::decode() の呼び出しが行われている。これを、下位レイヤに新設した、バイト列を受け取る CDR::get_octet_sequence() への呼び出し列に、意味を保ちつつ変換する等の軽量化を施した。

## 実験環境

実験環境として、100baseTX で接続された Sun Ultral creator ワークステーション (CPU 166MHz, Solaris 2.5.1) を 2 台用いた。クライアントからサーバーのオペレーションを呼び出し、戻ってくるまでのラウンドトリップタイムを、gettimeofday(3C) コールで実測した。プロトコルには IIOP を用いた。実験に使用したオペレーションの IDL 記述を図 1に示す。

比較対象として、socket を直接用いて IIOP プロトコル処理を全て手書きしたテストプログラムを準備した。このプログラムの実行時間が、処理速度の軽量化限界と考えられる。

## 実験 1. ORB コアの軽量化

ORB コア内部での処理の軽量化を確認するため、引数の少ない場合について実験を行い、第 3 節に示した設計指針 1 ～ 4 の影響を見る。1 台のホスト上でのローカル通信、及び、2 台のホスト間で、set, get オペレーションを繰り返し呼び出し、平均所要時間を測定する。

参考のために、IIOP を用いる市販 ORB 製品 A での測定も行った。プロトタイプ版は前述のようにエラーチェック処理等が不十分のため、製品 A とは対等の比較にはなっていない。

結果を図 2に示す。図中、軽量化前が DONA(org), 軽量化後が DONA(opt) である。

この結果から、local な通信の場合には、socket を直接使用 (= 最適値) の 1.8 倍の処理時間から 1.5 倍の処理時間へと軽量化されたことが分かる。残りの 0.5 倍分

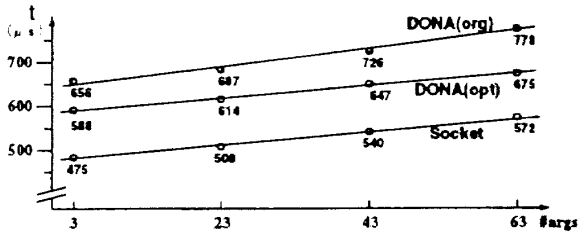


図 3: 引数を増やした場合の平均実行時間

の処理時間の主な要因は、軽量化後の処理にも多数残っているクラス継承・動的メモリ確保処理と推測される。軽量な ORB を設計する際は、これらを極限まで削減することが重要であることが分かった。

## 実験 2. stub, skeleton の軽量化

交換ノードシステムへの応用では、多数の引数を持つオペレーションが頻繁に現れる。そこで、第 3 節に示した設計指針 1 の適用により、stub, skeleton 内部で行われる引数処理がどの程度軽量化されるかを見る。

引数の数を 3, 23, 43, 63 個と変化させて、2 台のホスト間での実行時間を測定した。グラフの傾きが 1 引数あたりの処理時間に相当する。結果を図 3に示す。

この結果では、軽量化前のプロトタイプ版は、socket を使用した場合に較べ処理時間の増加度合いが 2 割程度大きくなっている。軽量化後は、socket 使用の場合とグラフの傾きがほぼ同一、すなわち引数処理に要する時間が等しくなっており、軽量化の効果が確認できた。

## 5　おわりに

本稿では通信ネットワークアプリケーションに適した ORB の処理方式として、軽量な ORB を設計するための指針を示した。その指針を適用して既存 ORB の軽量化を行った実験の結果、ORB コア・stub・skeleton ともに、一定の軽量化を達成することができた。本稿の検討結果は、我々が現在開発中の DONA DPE カーネル プロダクト版に反映させていく。今後は ORB 内部のエラーチェック処理についても、本稿の軽量化指針を適用していくことが課題である。

## 謝辞

プロトタイプ版の所要時間解析および Linux 移植にご尽力いただいた NTT ネットワークサービスシステム研究所の石谷和久氏に感謝します。

## 参考文献

[1] OMG. The Common Object Request Broker: Architecture and Specification.
ftp://ftp.omg.org/pub/docs/formal/98-02-01.pdf.gz.

[2] 武本 充治 ほか. マルチスレッドを用いた分散オブジェクトの実現方式. 情処全大 *(3) 4H-05*, pages 543-544, 1998.

[3] Sun. Inter-ORB Engine version 1.1.
ftp://ftp.omg.org/pub/contrib/interop/.