インタフェースマッチング技術の分散オブジェクトへの適用[1]

5J－6

NTT 情報通信研究所 樋渡仁

1 はじめに

近年、ActiveX を代表とするソフトウェアコンポーネントが実用に供されている。現在、コンポーネント数は増加する傾向にあり、特定の機能を実装するコンポーネントが複数存在する状況が到来しつつある。通常、同じ機能を実装するコンポーネントであってもインタフェースは異なるため、機能は変更せずにコンポーネントやコンポーネント数を変更したいという要請を満たすためには、利用者がインタフェースを一致させるラッパコードを記述しなければならないという課題があった。

本稿では、ラッパコードを自動生成するインタフェースマッチング技術について述べ、Java 言語による実装を評価する。そして、分散オブジェクト技術 Java RMI(Remote Method Invocation)への適用事例を報告する。

2 インタフェースマッチング技術

インタフェースマッチング技術とは、クライアントが要請するインタフェース記述（クライアント側）とコンポーネントが実装するインタフェース記述（サーバ側）が不一致の場合（図 1 上段）に、双方の記述からコンポーネントを集約するラッパコードを自動生成（図 1 下段）する技術である。

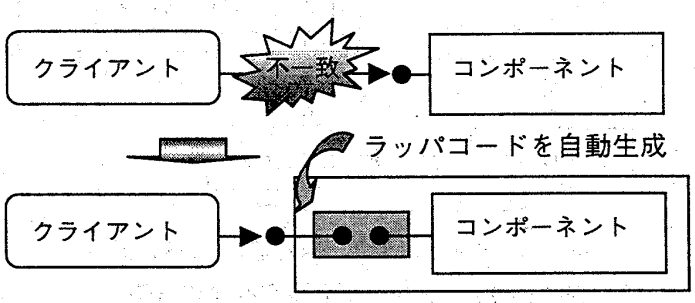

図 1:インタフェースマッチング技術の原理

2.1 インタフェース定義

本稿では、インタフェースを下記のように定義する。

- インタフェース：メソッド 1,...,メソッド N
- メソッド：返却型 メソッド名（引数型 1 引数名 1,...,引数型 M 引数名 M）

2.2 インタフェースマッチングアルゴリズム

インタフェースマッチング技術において、クライアント側各メソッドに対応するメソッドがサーバ側に存在する状況を対象としたアルゴリズムを以下に示す。ただし、対応するとは、双方のメソッドにおいて、引数数及び返値型が一致し、引数型が全て対応している場合を表現するものとする。

(1) クライアント側とサーバ側インタフェース記述から、各インタフェースに含まれるメソッド及び各メソッドに含まれる引数を抽出
(2) サーバ側インタフェースが複数の場合には、サーバ側インタフェースに含まれるメソッドはサーバ側メソッドとして一括して認識
(3) クライアント側メソッドがサーバ側では、どのメソッドに対応しているかを抽出
(4) 全部の組み合わせを調べて、クライアント側メソッドが複数のサーバ側メソッドが対応する場合には、利用者が選択
(5) 対応する各メソッドにおいて、引数の対応関係を抽出
(6) クライアント側及びサーバ側のメソッド及び引数の対応関係を利用して、クライアント側インタフェースを実装する新たなコンポーネントを定義
(7) その内部に再利用するコンポーネントを集約して、インタフェース側メソッドが起動されると、メソッド及び引数の対応関係から適切にサーバ側のメソッドが起動されるコードを自動生成

3 実装

Java 言語において、クライアント側インタフェース記述（図 2）及びサーバ側インタフェース記述（図 3）を入力し、ラッパコード（図 4）を自動生成するツールを実装した。本ツールにより、クライアント側及びサーバ側インタフェース記述におけるメソッド順序や引数順序といった不一致が解消されることが確認できた。

なお、マシン環境は Sun SS10, 40MHz, 128MB、OS 環境は Solaris2.5.1、Java 環境は JDK1.1.3、Java パーサは JavaCUP0.10g を利用した。

```
interface Iclient {
  Integer getID(String name);
  void putID(String name, Integer ID);
}
```

図 2:クライアント側インタフェース記述

```
interface Iserver {
  void putIt(Integer It, String index);
  Integer getIt(String index);
}
```

図 3:サーバ側インタフェース記述

[1] A Study of Applying Interface Matching Technology to Distributed Objects
Jin Hiwatashi, mailto:hiwatasi@isl.ntt.co.jp
NTT Information and Communication Systems Laboratories

```
class IclientImpl implements Iclient {
   IserverImpl p = new IserverImpl();
   public Integer getID(String name) {
     return p.getIt(name);
   }
   public void putID(String name, Integer ID) {
     p.putIt(ID, name);
   }
}
```

図 4:自動生成されたラッパコード

4 評価

本技術の実用的な適用領域を見極めるために、Java クラスライブラリ (java.*) に含まれるクラス定義[1]において、引数数及びメソッド数の最大値を調査した結果（最大引数数:6、最大メソッド数:75）から、引数数は 1 から 7、メソッド数は 1 から 100 を適用領域として設定した。一方、実行時間に関しては、ウィザード[2]作成を考慮して、60 秒以下を要求条件として評価した。

4.1 メソッド数増加時の実行時間に関する評価

クライアント側及びサーバ側でメソッド数を 1 及び 10 間隔で 10 から 100 まで増加させた時にラッパコードを自動生成するのに要する実行時間を計測した結果を図 5 に示す。なお、各メソッドが持つ引数数は、2, 4, 6 の各条件で計測した。

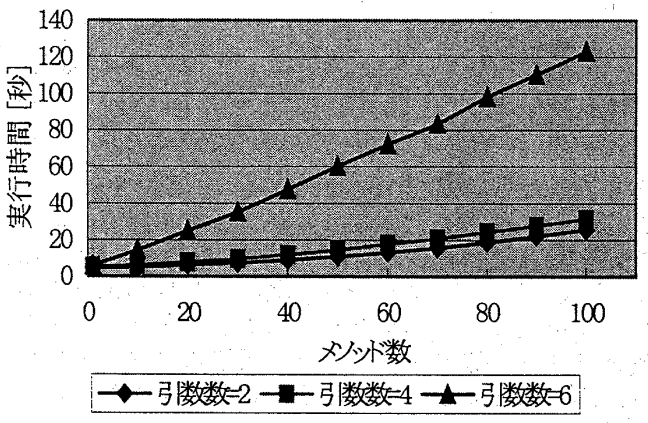

図 5:メソッド数と実行時間の関係

本評価により、引数数:6、メソッド数:50 以上では、実行時間に関する要求条件が満たせないという課題が顕在化したが、メソッド数の増加に対して、実行時間も線形に増加するため、計算機高速化により本課題は解決できる。

4.2 引数数増加時の実行時間に関する評価

クライアント側及びサーバ側メソッドが持つ引数数を 1 から 7 まで増加させた時にラッパコードを自動生成するのに要する実行時間を計測した結果を図 6 に示す。なお、各メソッド数は、10, 20, 30 の各条件で計測した。

本評価により、引数数が 7 を越えると実行時間に関する要求条件を満たせないことが明らかになった。しかし、実用的な適用領域においては、引数数が 6 を越えることは少ないため、実用上問題はない。

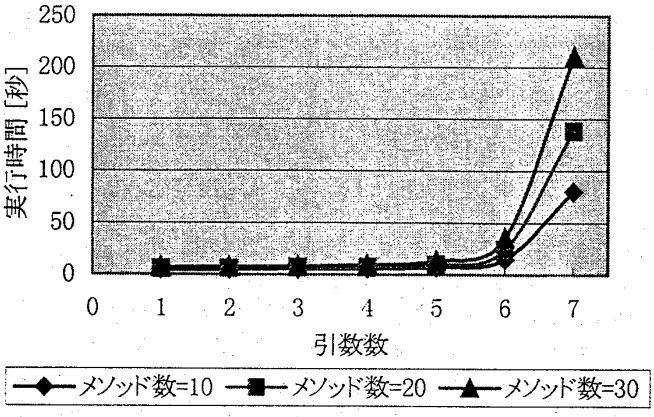

図 6:引数数と実行時間の関係

5 適用事例

5.1 分散オブジェクト技術 Java RMI への適用

Java RMI を代表とする分散オブジェクト技術は、リモートホスト上に存在するオブジェクトのメソッドを起動する技術である。図 7 に示すように、分散オブジェクトにおける開発工程には、クライアント側インタフェースからスケルトンとスタブを自動生成する機能が存在する。サーバ側機能が既存コンポーネントで実装されている状況で、この開発工程にインタフェースマッチング技術を適用すると、サーバ側インタフェース記述からサーバ側ラッパコードも自動生成できる。

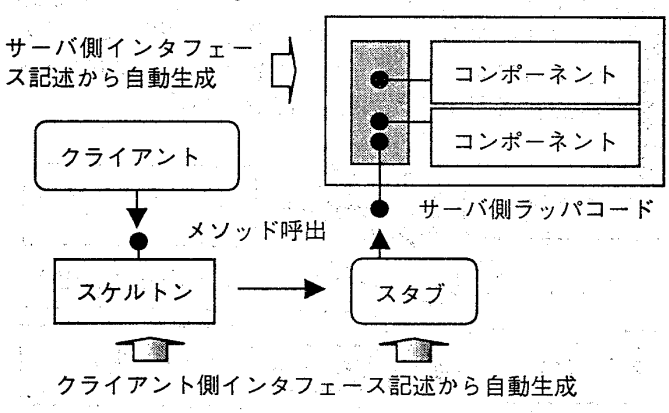

図 7:分散オブジェクト技術へ適用事例

評価用に実装したツールを Java RMI 用に改造、サーバ側コードを自動生成し、正常に動作することを確認した。

6 まとめ

本稿では、インタフェースマッチング技術及び本技術を実現するアルゴリズムを提案し、評価によって実用的な技術であることを示した。また、分散オブジェクト技術とも親和性が高く、僅かな変更で CORBA や COM にも適用できる見通しを得た。

参考文献

[1] JAVA クイックリファレンス, デビッドフラナガン, オライリージャパン (1997).

[2] T. Oren, et al., Guides: Characterizing the Interface, in B. Laurel ed., The Art of Human Computer Interface Design, Addison-Wesley (1990).