分散オブジェクト環境における参照機構の拡張

岡本 雅幸[1]・納富一宏[2]・石井博章[2]

1 博士前期課程 情報工学専攻
2 情報工学科

Extension of reference on distributed object environment

Noriyuki OKAMOTO[1)], Kazuhiro NOTOMI[2)], Hiroaki ISHII[2)]

Abstract

In this paper, we suggest an extension of reference between remote objects on distributed environment. Using generally remote reference methods, we had problems at object serialization and cannot keep on running programs when server environment were changed. Because informations about remote reference were managed by client objects.

For clear this limitation, we attempted to extends remote reference system. In this extention, we use reference information server for managing reference informations between remote objects and managing server environments. Client objects can get server information from reference information server. As a result, distributed system with this method were more safe and flexible.

Key Words: Distributed Object Environment, Remote Reference, Software Development Environment

1 はしがき

分散環境において、遠隔オブジェクトを利用するプログラムを容易に記述するために、遠隔オブジェクトをあたかも同一ホスト上にあるかのように使うことのできる「遠隔参照」は最も基本的な技術である。

本稿で対象とするJava言語においては、標準APIとしてRMI(Remote Method Invocation)という仕組みが用意され、遠隔参照を用いた分散オブジェクトを容易に使用することができる。しかし、遠隔オブジェクトへの参照はネットワークのコネクションに依存しており、何らかの原因によりネットワークの遮断やサーバーの障害等が起きた場合、プログラムの実行を継続する事は困難である。また、遠隔オブジェクトへの参照を保持しているオブジェクトを直列化する場合、参照元ホストと参照先ホストが復元後に異なる場合があるため、直列化／復元機構を安全に利用できるとは言えない。

そこで本稿では、従来のコネクション依存の方式とは異なる新たな遠隔参照の方式をを提案する。この方式では、オブジェクト間の参照を管理する参照情報管理サーバを用いオブジェクト間の参照情報および各サーバの状態を一元管理することにより、サーバ環境の変化の際にもプログラムの実行を継続することが可能となる。また、参照情報管理サーバがサーバ環境を一元的に管理するため、遠隔参照を保ったまま直列化されたオブジェクトを別ホストで正常に復元することも可能となる。

本稿では、この方式に基づく実装を行うと共にその結果を考察する。また、この拡張により生じる問題点についての検討を行う。

2 一般的な遠隔参照の手法

一般的に用いられている遠隔参照は、参照元／先オブジェクト間の関係だけで考えられる。この節では、オブジェクト間の遠隔参照として一般的に用いられている Java 言語上の RMI について述べる。

2.1 Java 言語での RMI

Java 言語上で標準 API として用意されている RMI では、オブジェクト間の参照はサーバ名とオブジェクト名の２つの要素によって決定される。参照はクライアントからサーバへ向けて作成され、クライアントは参照に関する操作 (参照の解消／再設定) を行うことができる。

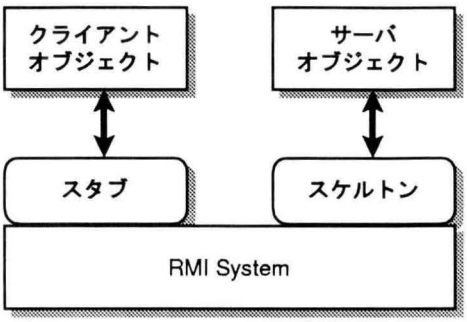

図 1: RMI システムの概要

RMI においては、クラスのコンパイルの際に「スタブ」「スケルトン」と呼ばれる２つの補助クラスを作成し、遠隔参照機構を実現している (図 1)。サーバオブジェクトに対するメソッド呼び出しはこの２つのオブジェクトを通して透過的に行なわれるため、クライアントオブジェクトからはサーバオブジェクトがあたかも同一ホスト上に存在するかのようなメソッド呼び出しが可能となる。クライアントからサーバへのの参照は、java.rmi.Naming クラスを通じて取得され、クライアント側からの操作を行うことが可能である。

2.2 従来の手法での問題点

2.1 節で示した RMI の手法では、クライアント環境の変化に対してはクライアント側のプログラムにより対処できるが、サーバ環境の変化には対処できない (図 2)。

遠隔参照はネットワークのコネクションによって確立されているため、内部で遠隔参照を保持しているオブジェクトをそのまま直列化することはできない。また、一旦遠隔参照を解消した後に直列化した場合でも、復元の際にサーバ環境が変化する可能性があるために安全に直列化できるとはいえない。分散環境において、ホスト間でのオブジェクトの移送を行うプログラムやオブジェクトの永続化を行なうプログラムの開発に際し、このことは大きな制約となる。

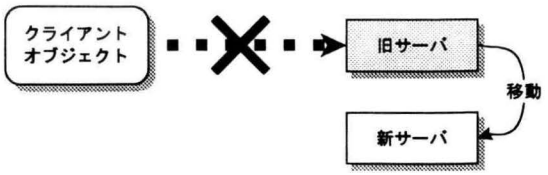

図 2: サーバ環境の変化に伴う不整合

2.3 システムの提案

そこで本稿では、異なるホストに存在するオブジェクト間の参照情報を管理する専用のサーバを設け、proxy オブジェクトを拡張することにより、遠隔参照を保ったままでのオブジェクトの直列化／復元が可能なシステムを提案する。また、参照情報を一元管理することによりサーバ環境の変化に対しても参照を正しく保持し、プログラムの実行を正常に継続することが可能なシステムの構築を実現する (図 3)。

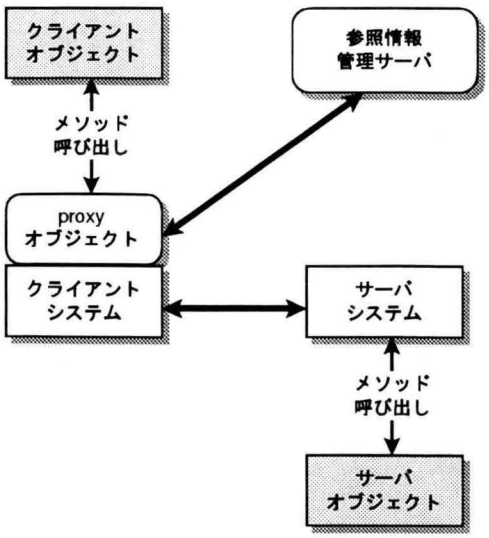

図 3: 本システムの概要図

システムは参照元オブジェクトが存在するクライアント、参照先オブジェクトが存在するサーバ、そして参照管理サーバから構成され、これらのサーバと拡張された proxy オブジェクトが互いに情報を交換することにより動作する。

3　本稿で提案するシステム

この節では、2.3 節で提案した拡張方式に基づいた設計について述べると共に、実装についての検討を行う。

3.1　参照情報管理サーバ

参照情報管理サーバは、クライアントオブジェクトからサーバオブジェクトへの参照情報およびサーバ環境に関する情報を管理／提供する。遠隔参照は実際には proxy オブジェクトからサーバオブジェクトへの参照である。参照情報管理サーバでは、proxy オブジェクト及びサーバオブジェクトに対して一意なオブジェクト ID を割り当て、この ID を用いて各参照情報を識別する。実際に管理される遠隔参照情報は、以下の通りである。

- proxy オブジェクトのオブジェクト ID
- サーバオブジェクトのオブジェクト ID
- proxy オブジェクトが存在するホスト情報
- サーバオブジェクトが存在するホスト情報
- 参照状態

上記以外に、各サーバ上に存在するオブジェクトのオブジェクト ID も一元管理される。サーバ環境の変化が検出されると、上記情報を元に各参照に関する情報を更新する。

3.2　proxy オブジェクトの直列化

proxy オブジェクトを直列化可能にするため、以下のような動作を行なうよう proxy オブジェクトを拡張する。

1. **メソッド呼び出し:** サーバオブジェクトのメソッド呼び出しが失敗した場合は、何らかの原因によりサーバ環境が変化したと考えられる。最新のサーバ環境を得るため、参照情報管理サーバへ問い合わせを行ない、その結果に従って参照情報を更新する。
2. **直列化:** 直列化を行なう際、サーバオブジェクトへのコネクションを一旦解消する。また、直列化の際には参照情報管理サーバへ情報を送信し、参照状態を変更する。直列化後のバイトストリームには、サーバオブジェクトが存在するホスト情報も記録される。
3. **復元:** 復元時には、記録されているサーバ情報を基にサーバへ接続を試みる。サーバへの接続が失敗した場合や、対象オブジェクトがサーバに存在しなかった場合は参照情報管理サーバへ問い合わせを行ない、サーバオブジェクトの実際の位置を得、新たにコネクションを確立する。同時に参照情報管理サーバ上の参照状態も更新される。

3.3　参照状態の遷移

本システムにおいて、proxy オブジェクトが保持するクライアント／サーバオブジェクト間の参照状態は図 4 のように遷移する。

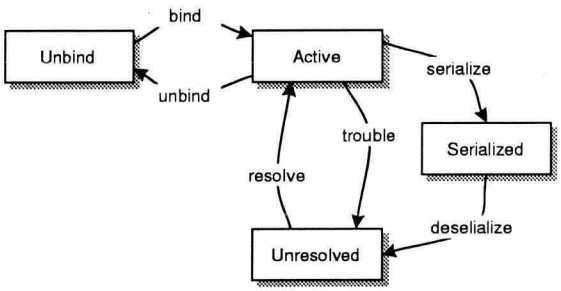

図 4: 参照状態の遷移図

Unbind オブジェクト間の参照がない状態。参照が作成される前の初期状態、およびサーバオブジェクトの使用が終った終了状態がこれに当たる。クライアントからの明示的な操作により **Active** 状態との遷移が起きる。

Active オブジェクト間の参照が正常に機能している状態。遠隔参照がこの状態にある場合に限り、遠隔メソッド呼び出しが可能である。

Serialized クライアントオブジェクトが直列化されている状態。この状態では、クライアントからサーバへの参照は存在するが、コネクションは途切れている。

Unresolved オブジェクト間の参照情報が正常ではない状態。復元直後の状態がこれに当たり、proxy オブジェクトが参照情報管理サーバへ問い合わせを行ない情報を得ることで **Active** 状態へと遷移する。

3.4 例外処理

図 4 において、unresolved 状態から active 状態への遷移は、保守やトラブルなどの何らかの原因によってサーバ環境が変化した際に起きる。この遷移の存在により、プログラムの実行が継続可能となる。サーバが利用できない場合はクライアントオブジェクトへ例外が送出される。

4 評価と考察

この節では、3 節で述べた仕様に基づき実際の実装を行い評価すると共に、このシステムの問題点について検討を行う。

4.1 実装

提案した方式に従い、システムの実装を行なった。実装には Java 言語を用いた。これは言語自体でオブジェクトの直列化に関する機能を有するからである。 また、Java 言語で実装を行うことにより、マルチプラットフォームを実現した。

遠隔参照を実現する ORB クラス、ORBServer クラスを作成した。また、参照情報管理サーバとして ORBLocator クラスを作成した。

4.2 評価

実装したシステムを用い、サンプルとしてごく簡単な遠隔参照を行なうプログラムを作成し、その動作を確認した。遠隔参照を取得し、サーバオブジェクトのメソッドを実行し結果を受け取るプログラムを図 5 に示す。また、実際の動作の状況を図 6 に示す。

図 5 のプログラムにおいて、RemoteHello はインタフェイスとして宣言され、サーバオブジェクトのメソッドを定義する。このインタフェイスを実装した RemoteHelloImpl クラスを作成し、ORBServer の管理の元でサーバオブジェクトとして動作させた。また、遠隔メソッド呼び出しを行なう proxy クラスとして RemoteHelloProxy クラスを作成し、クライアント側からはこれを利用し遠隔参照を実現した。

```
import java.net.*;
import java.io.*;
import jp.ac.kanagawa-it.ic.ish.mill.*

// クライアントクラス
public class Sample {
  RemoteHello hello;

  public Sample() {
    // サーバオブジェクト "hello" に対する
    // 遠隔参照を得る
    hello = new
        RemoteHelloProxy("serv");
  }

  public void sayHello() {
    // サーバオブジェクトの getWords
    // メソッドを呼び出し結果を受け取る
    String helloWords =
        hello.getWords();
    System.out.println(helloWords);
  }

  public static void main(
        String args[]) {
    Sample instance = new Sample();
    Sample.sayHello();
  }
}

// 遠隔オブジェクトが実装する
// インタフェイス
public interface RemoteHello {
  public String getWords();
}

// 遠隔オブジェクト
public class RemoteHelloImpl
  implements RemoteHello
{
  public String getWords() {
    return new String("Hello!");
  }
}
```

図 5: 評価に用いたプログラム

次に、クライアントオブジェクトを直列化し、別ホストにて復元を行ない、プログラムの実行が正しく継続される事を確認した。また、クライアントオブジェクトからの参照を保ったままサーバを停止し、別ホストにてサーバを再起動し、プログラムの実行が正常に継続される事を確認した。

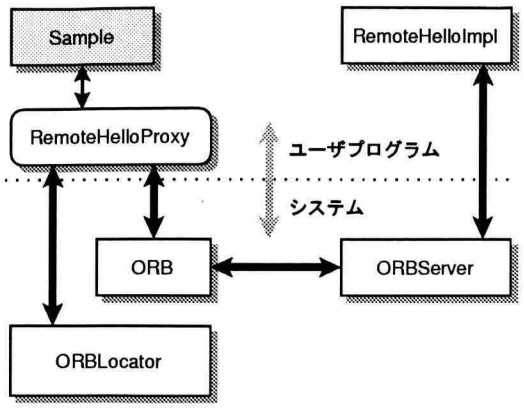

図 6: サンプルプログラムの動作

4.3 本システムを用いる際の制約

4.3.1 プログラムを作成する上での制約

プログラムを実際に作成する際、どのオブジェクトをサーバオブジェクトとして動作させるかを決定し、実際の操作をインタフェイスとして分離しなければならない。

4.3.2 proxy クラスの生成

proxy オブジェクトは、サーバオブジェクトがあたかもクライアントオブジェクトと同じホスト上に存在するかのように振舞う。これを実現するため、サーバクラスの操作をインタフェイスとして分離し、proxy クラスもこれを実装する。

proxy オブジェクトは、サーバオブジェクトの種類ごとに作成しなければならない。しかし、Java 言語では、クラス情報を動的に得ることのできる Reflection 機構が用意されているので、proxy クラスの自動生成は可能である。さらに、システム内に proxy クラスの生成機構を内蔵することにより、proxy オブジェクトの動的生成も可能となる。

4.3.3 参照情報管理サーバについての制約

本システムでは、全てのオブジェクト間の遠隔参照は参照情報管理サーバによって管理される。そのため、何らかの障害によりこのサーバが使用不能になってしまうと、全ての遠隔参照が使えなくなってしまう。

この問題に対応するため、サーバの２重化や分散管理など、対障害機構を備える必要がある。

5 むすび

本稿で提案した遠隔参照の拡張により、遠隔参照を用いたシステムがより柔軟に構築でき、かつ安全に利用できるようになった。

分散環境は、今後のソフトウエア技術の中核となるものであり、単一ホスト環境との差を縮めていく必要があると思われる。従来複雑であった分散環境上での開発に際し、その制約を可能な限り少なくしていくことが不可欠であるといえよう。

参考文献

[1] “Java Remote Method Invocation”, SunMicrosystems, http://java.sun.com/products/jdk/1.2/docs/guide/rmi/index.html(1999).

[2] “Object Serialization”, SunMicrosystems, http://java.sun.com/products/jdk/1.2/docs/guide/serialization/index.html(1999).

[3] 納富一宏, 岡本雅幸, 石井博章:”Java 言語によるオブジェクト入出力ストリームの拡張”, 神奈川工科大学研究報告,B-23,pp.91-96.(1999).

[4] Elliottte Rusty:”JAVA ネットワークプログラミング”, 株式会社ユニテック訳, 戸松豊和監訳,O’REILLY ジャパン (1997).