ソフトウェア工学 125－1
（2000．1．14）

モバイル環境における分散オブジェクトシステムの実現とその評価

山口 実靖　　野上 耕介　　相田 仁　　齊藤 忠夫

東京大学 工学部 電子情報工学科

分散オブジェクトシステムが発達し、ネットワークアプリケーションを短時間で作成が可能となった。しかし、既存の分散オブジェクトシステムにはモバイルコンピューティングなどの常時接続でない環境を十分に考慮に入れたものは少ない。本稿では非同期通信環境における分散オブジェクトシステムを提案、実装して、その評価を行った。提案手法では非接続状態における情報の伝達を考慮して冗長であるが情報を広く伝えられる方法を採用した。本手法の伝達能力をシミュレーションにより測定した結果、本手法は高い伝達能力をもち、特に通信相手に偏りがある劣悪な通信環境に強いことが示された。

Progress of Distributed Objects Systems in recent years enables developing Network Applications in short time. Distributed Object Systems are progressing in recent years. These systems ease Network Applications development procedure. However, existing Distributed Object Systems do not take mobile computing environment into account. We propose, implement and estimate Distributed Objects System in Asynchronous Communication Environment like mobile computing. The distribution method adopted in this system is redundant but enables fast spread of information. We adopted it because spread speed of information is very important in the environment. It was found with simulations that the proposed method has high capacity to spread information and it hardly depends on partiality of communications.

1　はじめに

本稿では非同期通信環境における分散オブジェクトシステムの提案を行い、その実装の評価と説明を行う。

ネットワークアプリケーションの開発を容易にする技術として分散オブジェクトシステムが注目され、近年非常に発展している。また、モバイルコンピューティングやPPP接続の普及により、常時接続でない環境(以後間欠接続環境と呼ぶ)における計算機の使用も重要となってきている。

しかし、現在の分散オブジェクトシステムは常時接続された環境を想定しており間欠接続環境での十分な使用はできない。

間欠接続環境では主に電子メールなどの非同期通信が行われる。TCP, UDP ポートを変えて FireWall を越えるために電子メールを媒体と使う分散オブジェクトシステム [1] や、メッセージを他のユーザに送信してから応答が返ってくるまでブロックせずに(待たずに)、処理を続けて次のメッセージの送信を行える非同期通信をサポートする分散オブジェクトシステムは存在するが、広い意味の非同期的通信(後述)環境をサポートする分散オブジェクトシステムはまだない。

また、ディスコネクティッドオペレーション、狭帯域 高遅延環境での処理に対する研究として成果を挙げているものも多いが([6], [7], [8], [9])、本稿はこれらのシステムより管理の分散および伝搬の速度で勝るものである。逆に本システムの方が資源の使用量が冗長となっている。接続環境のでの分散管理システムとしては文献 [10] 等がある。

ここで言う広い意味での同期的な通信とは通信遅延時間が一定以下であることが保証されている通信であり、非同期的通信とは保証されない通信である(文献 [2] 参照)。TCP コネクションを直接張って行う通信は(実際は遅延の保証はないが)本稿の要求では十分小さい時間で応答を得られることが期待できるので同期的通信として扱う。

第2章にて非同期通信環境における分散オブジェクトシステムの方式の提案を行う。第3章にて提案システムの定性的評価、情報伝搬の評価を行う。特にモバイルコンピューティング環境ではある特定の相手としか通信ができないことが多いことを考え通信相手の偏りによる弊害の大きさについて述べる。第4章にて提案システムの Java 言語による実装の説明 / 問題点を述べる。第5章にて本稿のまとめと今後の課題について述べる。

2　提案手法

本章では本稿の提案手法 “非同期分散オブジェクトシステム” について述べる。

2.1　想定している環境

モバイルコンピューティングなどのように通常はスタンドアローンな環境で使用しているが、まれに接続機会に恵まれる環境を想定している。また、接続は地理要因、コスト的要因に支配されることも多いため、接続は与えられるものであり、自由に作れるものではないとした。

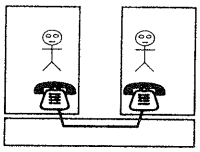


図 1: 一般的な分散オブジェクトシステムのモデル

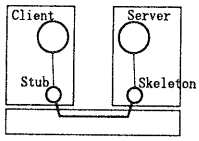


図 2: 一般的な分散オブジェクトシステム

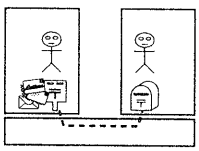


図 3: 間欠接続環境における分散オブジェクトシステムのモデル

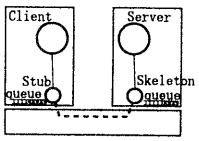


図 4: 間欠接続環境における分散オブジェクトシステム

2.2 提案手法の構造

一般の分散オブジェクトシステムは図 2の構造をしており、図 1の様なモデルで例えることができる。すなわち、サーバとクライアントは電話で繋がっており、電話をかけるとすぐに情報を伝えることが可能である。本稿で想定している環境は図 3の様なモデルである。すなわち、ユーザ同士は通信手段を持っているが送信した情報はいつ届くか分からない。図 4の様な構造をしている。送信情報はキューイングされ、いつか届くことが期待されるがその時間的保証はない[1]。

2.3 機能

以下に提案手法の機能を述べる。第 2.3.1章の 1は一般的な分散オブジェクトシステムで必要な機能であるが、2, 3, 4は非同期分散オブジェクト特有の機能である。

第 2.3.2章の機能は非同期通信環境であるため解決はできない。

2.3.1 要求される機能

非同期通信環境で動作するに以下の機能が必要となり、それを備えている

1. 非同期通信環境において他の計算機のサーバオブジェクトのメソッドを起動することが可能であり、その戻り値を獲得することも可能のである。
2. サーバオブジェクトのメソッドの起動要求を作成する瞬間、サーバオブジェクトが戻り値を返す瞬間に他の計算機と接続されている必要はない。
3. 常時稼動しているサーバマシンを必要としない
4. 任意の 2 計算機が同時に稼動している必要がない

[1]提案システムの証している環境は伝搬の遅延時間が定まっていないので、ほぼ一定時間で情報が届く郵便とは厳密には異なっている

2.3.2 実現されない機能

提案手法では以下の機能は実現されない

1. 有限時間以内に、メソッド起動要求をサーバに伝える / 戻り値を得る
2. メソッド機動要求、戻り値が届いたか否かの確認をする。
3. サーバがアドレスを分かっていないクライアントからのメソッド呼び出しを受ける

機能 3は通常の分散オブジェクトでは実現されている。TCP/IP を用いた分散オブジェクトシステムではクライアントがサーバに対してコネクションを確立すれば双方向通信が可能であるため戻り値を返すことが可能であるが、非同期通信ではサーバがクライアントの位置を知っていなければ戻り値を返すことができない。

2.4 伝達情報

ユーザ S からユーザ R へのメッセージ - 提案手法において、ユーザ S からユーザ R への伝達情報は以下のものがある。

1. ユーザ S による、ユーザ R へのメソッド起動要求
2. ユーザ R による、ユーザ S へのメソッド起動要求の戻り値

通信方法 - 非接続中にユーザ S がユーザ R への情報伝達要求を発生させたら

1. ユーザ S は、その内容をユーザ R への送信キューに入れる
2. ユーザ S が、ユーザ X(R と別のユーザでも良い)へ情報送信の機会に恵まれたらそのキューをユーザ X に送信する

目的のユーザ以外のユーザに伝えることの意義は第 2.6章で示す。

2.5 遠隔メソッド起動方法

メソッド起動要求 (クライアント側) - その際 通常の分散オブジェクトシステムでは、クライアントはシステム (の提供する代理人[2]) に対してメソッド要求をだし、システムがその要求を目的のサーバに送信する。提案システムでも同様にクライアントはシステム (の提供する代理人) に対してメソッド起動要求を出す。このメソッド起動要求はシステムによりキューに入れられ、送信可能時に送信される。また、後に戻り値が得られたときに行う処理を登録しておく。

メソッド起動 (サーバ側) - ユーザ C からユーザ S へのメソッド起動要求がユーザ S にどといたら、ユーザ S はそれを実行し、その戻り値をユーザ C へのメッセージキューにいれる。

[2]一般に Proxy, Stub と呼ばれる

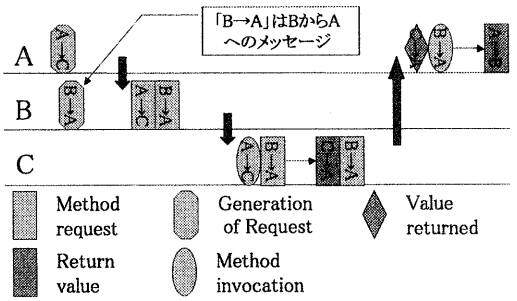


図 5: 伝搬例

戻り値の獲得 (クライアント側) - 戻り値が得られたら、(要求時に) 登録した処理を実行する。その際、その処理の引数として得られた戻り値を用いる。

2.6 伝搬例

図 5にキューの伝搬例を示す。図においてユーザ A は C に対するメソッド起動要求を作成する。B は A に対する起動要求を作成する。ここで A は B に対して情報を送る機会があり、自分 (A) のキューを B に送る[3]。これにより B のキューは「A → C(起動要求)」「B → A(起動要求)」の 2 個になる。次に、B は C に情報を送る機会があり、B のキューは A にコピーされる。すなわち C のキューは「A → C(起動要求)」「B → A(起動要求)」となり、「A → C(起動要求)」は、目的のユーザ C に届いたので実行され「C → A(戻り値)」に変わる。C のキューは「B → A(起動要求)」「C → A(戻り値)」となる。最後に、C は A に情報を送る機会に恵まれ「B → A(起動要求)」「C → A(戻り値)」が A に届く。「C → A(戻り値)」は目的の A に届いたので "登録しておいた処理" をこの戻り値を引数に用いて実行する。「B → A(起動要求)」も目的の A に届いたので「A → B(戻り値)」に変わる。

この例のように直接接続していないユーザ同士でも情報を交換できていることが分かる。

3 評価

3.1 定性評価

提案方式によりモバイル端末などの間欠接続環境においてオブジェクト指向アプリケーションの作成を実現した。しかし、本システムは非同期通信環境の上に乗っておりスタンドアローンのオブジェクト指向システムや静的ネットワークで繋がった分散オブジェクトシステムとは以下の点で大きくことなる。

1. 戻り値の獲得が非常に遅く実際には戻り値の有効利用が難しい
2. メッセージ順序の保たれ方が弱い。同一ユーザが作成したメッセージの順序は保たれるが異なるユーザの作成したメッセージの順序は保たれない。

[3] C 宛てのメッセージであり、B 宛てのメッセージでないが B に対しても送る

また、本システムは完全に分散管理を行っているため中央のサーバを必要としていない。一台で管理することに比べて、各ユーザそれぞれで管理することは合計すると多くの資源を使っていることになり、冗長であるがそのときに稼働しているユーザ同士だけでも処理を続けることが可能であり、サーバと接続できないユーザが処理を行えないと言うこともない。このようにトレードオフの関係にあるが間欠接続環境には本システムが向いていると思われる。

3.2 定量評価

提案方式では、あるユーザへの送信情報を別のユーザにも投機的に送っている。これにより以下のトレードオフが成り立つ。

1. 間接的な伝搬[4]が可能となり、直接通信していないユーザに対しても情報を送信できる。
2. 理論上で最も早い情報の送信を可能にする。
3. 無駄な情報送信が多くなる

本章で間接伝搬の伝搬速度への効果について考察する。通信相手はランダムに選択するがその確率密度関数をとして (1) 図 9のように通信相手の選択に偏りがない場合を (第 3.2.2章), (2) 図 10のように通信相手の選択に偏りがある場合を (第 3.2.3章), (3) 図 11のように自分の近くのユーザしか通信相手に選ばない場合 (第 3.2.4章), (4) 図 12のように自分の近くのユーザしか通信相手に選ばず、しかも通信に偏りがある場合 (第 3.2.5章) の 4 通りを試した。

第 3.2.6章は間接伝搬ありとなしの未伝搬情報数の差を、第 3.2.7章で通信偏りありとなしが通信偏により受ける影響の大きさの差を述べる。

3.2.1 評価値とシミュレーション条件

シミュレーションにより後述の未伝搬情報数を測定しそれにより伝搬速度を評価した。

任意のユーザが別のユーザに対して情報送信要求を発生させたとき、その要求がすでに目的のユーザに届いていた場合 "伝搬済み情報" と呼び、まだ目的のユーザに届いていない場合 "未伝搬情報" と呼ぶ。

全てのユーザは単位時間に任意のユーザに対して 1 個の送信要求を発生させる。全てのユーザは単位時間に 1 回 1 ユーザに対してのみ情報送信の機会を与えられる。

時間の経過ともに要求が発生し、それに伴い "伝搬済み情報" は単調に増加する。しかも、定常状態において単位時間あたりの "伝搬済み情報" の増加速度は、単位時間あたりの要求の発生速度と等しいため定常状態における "伝搬済み情報" の比較はあまり意味がない。

"未伝搬情報" の数は定常状態においてある値に収束する。定常状態における "未伝搬情報" の数を評価値とする。

3.2.2 通信に偏りがない場合

通信ユーザは一様分布にランダムに選ぶ。シミュレーションを行い、間接伝搬ありとなしの定常状態での未伝搬情報数として図 6を得た。

[4] 又聞き

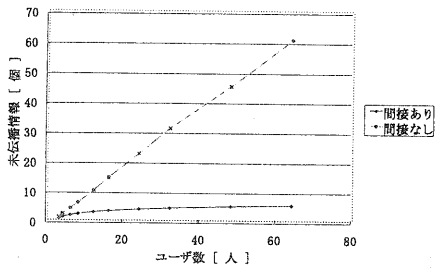


図 6: 通信相手に偏りがない

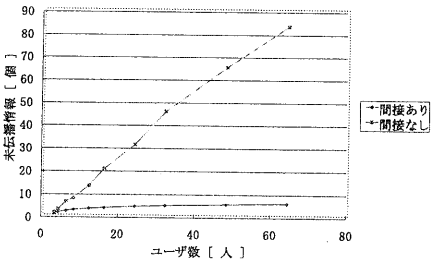


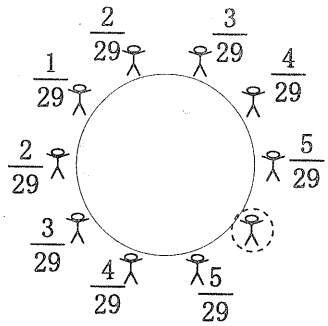
図 7: 通信相手に偏りがある

3.2.3 通信に偏りがある場合

各ユーザには連続したユーザ ID が割り振ってあるとする。自 ID と相手 ID の差により、通信相手として選ぶ確率は図 10のように変化する。ただし、ID はループしておりユーザ ID 最小とユーザ ID 最大は隣とした。例えば、ユーザ 10 人 (自分を除いて 9 人) の例では通信確率は図 8の様になる。このモデルではユーザ ID が自分と近い人を優先的に通信相手に選択する。第 3.2.2章と同様に間接伝搬ありとなしの定常状態での未伝搬情報数として図 7を得た。

3.2.4 通信にさらに偏りがある場合 (1)

第 3.2.3章よりさらに偏りがある場合として図 11のようにユーザ ID が自分と近いユーザしか選ばないモデルを用いてシミュレーションを行った。同様に図 14を得た。間接伝搬なしはユーザ ID の遠いユーザとは全く情報交換しないためこのモデルでは未伝搬情報数は発散し、シミュレーションは行えない。



右下のユーザが通信相手として選ぶ確率は最も近いユーザが $\frac{5}{29}$、最も遠いユーザが $\frac{1}{29}$

図 8: 通信偏り例

3.2.5 通信にさらに偏りがある場合 (2)

図 12のようにユーザ ID の自分と近いユーザしか選ばないモデルを用いてシミュレーションを行った。同様に図 13を得た。このモデルも間接伝搬なしはシミュレーションは行えない。

3.2.6 間接伝搬ありと間接伝搬なしの未伝搬情報数

通信相手の選択に偏りがない場合 (第 3.2.2章)、間接伝搬ありの未伝搬情報数はほぼ log(ユーザ数) に比例し、間接伝搬なしのそれはほぼユーザ数に比例している。図 6,7より、間接伝搬は十分大きな効果があることが示された。

3.2.7 通信偏への依存度の比較

間接伝搬を行わない方式では図 13の様に通信が偏ると情報の伝搬量が著しく減少する。それに対して、間接伝搬を行うと図 14のようになる。すなわち図 10の様な通信の偏りの影響をほとんどうけない。図 11,12の様な偏りの影響はかなり受けるが、実用が可能と思われる範囲の影響である。

4 実装

4.1 Java

本実装は Java 言語によって行われた。Java 言語 (JDK1.1[5]以降) を採用したのは以下の理由による。

1. Serialazation 機能が備わっている
2. Reflection 機能がの備わっている

4.2 モデル

本実装は図 15の様なモデルでメソッド呼び出しを行う。

[5] Java Development Kit

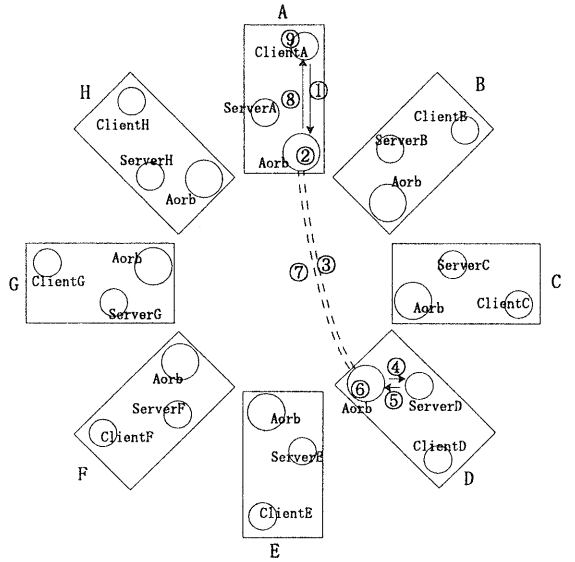


図 15: 提案システムのモデル

4.3 メソッドの転送

どのメソッドが起動されたかを記録しておく必要があるが、方法は以下の 2 種類考えられる。

1. メソッドの通し番号のみを記録しておく
2. メソッドシグニチャ (メソッド名, 引数の型) を記録しておく

方法 1は必要とする記憶領域が非常に少ないが、クラスファイルに変更が施されメソッドの通し番号にずれが発生すると正しく動作しなくなる。RMI, CORBA, HORB などの多くの分散オブジェクトシステムが採用している。

方法 2は冗長であるが、クラスファイルに変更が施されてメソッドの通し番号にずれが発生してもそのシグニチャを持つメソッドが削除されない限り正しく動作することが期待できる。

非接続状態ではユーザの知らないところで、他のユーザによってアプリケーションがバージョンアップされることがあることを考慮して冗長であるがより可用性の高い方法 2を採用した。

4.4 引数インスタンスの受け渡し

一般に引数の受け渡しには (1)pass-by-value, (2)pass-by-reference の 2 種類の方法が考えられる。計算機 (プロセス) を超えるインスタンスの受渡しでは特にこの問題が重要となる。

pass by value(値渡し) インスタンスのコピーを転送する方式。実引数と仮引数が別のインスタンスとなり、本来の Java のメソッド呼び出しとは異なる呼び出し方式。多くの Java 上の分散オブジェクトシステムがこの方式を採用している。

pass by reference(参照渡し) インスタンスの参照を転送する方式。仮引数は実引数への参照となり、同じインスタンスを参照しており、本来の Java のメソッド呼び出しと同じ。CORBA[3, 4] が採用している。

モバイル環境においては通信できない状態にあることが多いため、オブジェクトへの参照は可用性を著しく低下させる。よって、提案方式では pass-by-value を採用した。

また、コピーには深いコピーと浅いコピー [5] がある。浅いコピーの方が転送情報が少なくなるが、同様に間欠接続環境に置いて参照を辿れないことを考えて深いコピーを採用した。すなわち、参照先のオブジェクト全てがコピーされる。

4.5 戻り値の処理

メソッドを起動したとき、後に戻り値を獲得できたときの処理を登録しておく。その方法は ReturnListener インターフェイスを実装したクラスのインスタンスを指定することにより行う。ReturnListener を実装したクラスは

```
public void valueReturned
  ( java.io.Serializable returnValue);
```

を必ず持っていて、戻り値獲得時にはこのメソッドが獲得した戻り値を引数として実行される。

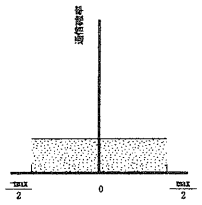


図 9: 通信相手 偏りなし

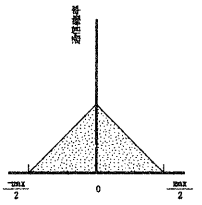


図 10: 通信相手 偏り三角

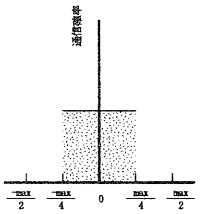


図 11: 通信相手 偏り半四角

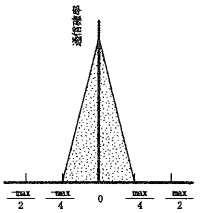


図 12: 通信相手 偏り半三角

上記は通信相手の選択の確率密度関数。横軸は自 ID と相手 ID の差であり、 max はユーザ数。自 ID と相手 ID の差は $-\frac{max}{2}$ から $-\frac{max}{2}$ の間となる。

4.6 変更の通知

通信が可能になったときに保存されているログを他のユーザに伝える必要がある。一般の分散オブジェクトシステムは TCP/IP 上になどに実装した通信システムを備えているが、本稿では電子メール、記録メディア、IrDA などの非常に疎な結合上でも動作することを前提としているため、通信システムは実装していない。ログをシリアライズするメソッドを機動することにより、転送すべきバイト列が戻り値として得ることができ、得られたバイト列を何らかの通信手段で他のユーザに伝える。通信手段に依存しないので常時接続を必要としないことはもちろん、非同期通信による転送も可能となっている。

逆に他のユーザからシリアライズされたログを受け取ったユーザはそれを自分のログに反映させる。まず、受け取ったバイト列をデシリアライズしてオブジェクトの配列に復元し、そして、その履歴の中で自分宛てのメッセージが存在したらそれを処理する。

さらに、自分の保持している伝搬状況テーブルを更新し、ある変更が消滅させてよいのなら消滅させる。オブジェクトの消滅については第 4.7章参照。

4.7 伝搬管理

各ユーザは自分宛て以外のメッセージも保持している。もし、このメッセージが目的のユーザに正しく届いたことが確認できたら、このメッセージは削除して良い。各ユーザは “あるユーザが別のあるユーザの作成したメッセージをどの時点まで知っているか” のテーブル (サイズ $O(n^2)$) を保持している。この情報は単調増加のため、

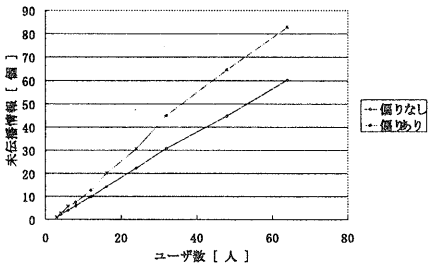


図 13: 間接伝搬なしの通信偏りの影響

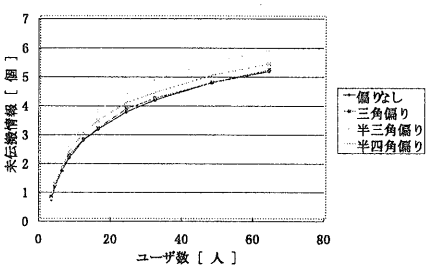


図 14: 間接伝搬ありの通信偏りの影響

大きな値が必ずより新しい情報 (すなわちより真実に近い) 情報となる。ユーザはメッセージ情報と共にこの伝搬テーブルの情報も交換している。テーブルの更新は次の 2 つの機会に行われる。

(1) あるユーザが新しいメッセージを受け取り、テーブルの自分の項目を更新する。

(2) 他のユーザからテーブルを受け取り自分のテーブルよりも新しい部分を更新する。

ここで、

(3) あるユーザに対して、メッセージを送信したので、テーブルのそのユーザの項目を更新する。

が考えられるが、非同期通信環境では以下のことが起こり得るので (3) では更新することができない。ユーザ S がユーザ R に対してユーザ R 宛てのメッセージ M を送信し[6]、その後ユーザ S はユーザ X に出会い、ユーザ R にはすでにメッセージ M を送った事を伝える。そこでユーザ X はメッセージ M を削除してしまうが、実際に

[6] ユーザが S が送信したのは確かだが、ユーザ R が受信したことは保証されていない

はメッセージ M がまだ届いておらず(いつ届くか保証されない)、逆にユーザ X の方が先に情報を伝えることが可能となってしまう。

また、テーブルは効率良く不要メッセージを削除するためのものであり、このテーブルが最適に動作しなくても通信量、使用メモリが冗長になるだけでユーザに届くメッセージが減ることはない。逆にテーブルが大きすぎて性能をそこなう場合にはこのテーブルの情報量を減らすことも可能である。

4.8 実装上の課題

1. 型チェックが実行時まで行われない。メソッドの起動を実行時に reflection を用いて行っているので receiver クラスのインターフェイスのチェック, 引数と戻り値の型チェックが実行時まで行われない(しかも実行速度が遅い)。ただし、これれにより(開発時になかった) 実行時に増えた新しいインターフェイスにアクセスすることができる。

2. 既存のコードの再利用が(そのままでは) 行えない。必ず ReturnListener を引数に付け加えなくてはならないので提案システム用に新しく実装したコードなら問題ないが、既存のコードは再利用できないことになる。これを回避する方法として Thread で別の処理の流れを作るなどしてメインの流れとは別の場所で戻り値処理 (登録と受け取り) を登録する方法やトランスレータを作成るなどが考えられる。

3. メソッド呼び出しを代理人が呼び出すので元来同一パッケージ内からのアクセスであったメソッドコールが異なるパッケージからの呼び出しとなってしまう。これよにり、Java のデフォルトアクセス制限 (Package 制限) のフィールド、メソッドにアクセスできなくなる。

4. 引数オブジェクトが参照渡し (pass by reference) ではなく、値渡し (pass by value) である。また、引数、戻り値のオブジェクトは Serializable を実装している必要がある。

5. それぞれ別の Virtual Machine で動作するため、static 変数 などが正しく共有できない。

5 まとめ

本稿ではモバイルコンピューティングなどの非同期通信環境に置ける分散オブジェクトシステムの提案を行い、その評価を行った。

提案方式により非常に疎な接続の環境における分散オブジェクトシステムを実現した(「同期通信を必要としない」「常時稼動しているサーバマシンを必要としない」「直接的通信を必要としない」)。第 2章で示したように直接通信を行えなくてもメソッド起動要求や戻り値を伝えることが可能であるが、交換する情報の冗長性が増している。モバイルコンピューティングなどの非同期通信環境で通信回数、通信相手に制限がある場合は 1 回の通信でできるだけ多くの情報を伝える本方式が有用となる。システムは伝搬するメッセージをバイト列に変換するだけなので、情報交換手段が E-Mail や交換可能メディアしかなくてもバイト列を交換できる環境では動作させることができる。また、提案システムの間接伝搬はオーバヘッドを増すが伝搬速度を大きく向上させ、通信の偏りの影響も受けづらいことを示した。

参考文献

[1] 藤崎 智宏, 浜田雅樹, "電子メールによる分散オブジェクトメッセージ交換機構", 情報処理学会 「分散システム / インターネット運用技術」研究会 第四研究会 4-9, 1996 年 3 月.

[2] 白鳥 則郎, 滝沢 誠: "分散処理", 丸善株式会社, 1996.

[3] http://www.corba.org/

[4] http://www.omg.org/

[5] 青柳 達也, "Java API プログラミング", 工学図書, 1996.

[6] Anthony D.Joseph, Joshua A. Tauber and M. Frans Kaashoek, "Mobile Computing with the Rover Toolkit", *IEEE Transactions on Computing*, pages 337-352, March 1997.

[7] David A. Nichols, Pavel Curtis, Michael Dixon and John Lamping, "High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System", *in Proceedings of ACM Symposium on UIST '95 Pitt. PA.*, November 1995.

[8] DouglasB.Terry他, "Managing Update Conflicts in Bayou, a Weakly Connected Replicated Strorage System", December 1995. availableat `http://www.parc.xerox.com/csl/projects/bayou/pubs/sosp-95/BayouConflictsSOSPPreprint.ps`

[9] `http://ficus-www.cs.ucla.edu/ficus/`

[10] Louis Thomas, Sean Suchter, Adam Rifkin "Developing Peer-to-Peer Application on the Internet : the Distributed Editor, SimulEdit"California Institute of Technology

A API

A.1 基本 API

Aorb(Object object, int users, int me)

システムのコンストラクタ。公開するサーバオブジェクトを保持。純粋なクライアントとして参加する場合はサーバオブジェクトは null でよい。共有するユーザ数, 自分の ID を獲得 (現在これらは実行時に変更できない)。

Object e(String methodName, Object[] argv, Class[] argTypes, int toUser, ReturnListener returnListener)

methodName という名前で、引数の型が argTypes である メソッドを引数 args で起動する要求を作成してキューに入れる。戻り値が戻ってきたときに行う処理を登録しておく。プリミティブ型はラッパーでラップす

る。toUser はメソッドを送信する相手の ID。

Object e(String methodName, Object[] argv, int toUser, ReturnListener returnListener)
methodName という名前のメソッドを 引数 args で起動する要求を作成してキューに入れる。戻り値が戻ってきたときに行う処理を登録しておく。メソッドの引数の型は 引数から予測する。プリミティブ型はラッパーでラップする。toUser はメソッドを送信する相手の ID。
予想が失敗する場合：プリミティブ型が引数に混ざっている時。ラップされたものなのか、されてないものなのかの区別つかない。

byte[] send()
自分の保持しているキューをバイト列に変換する。このバイト列を何らかの方法 (ORB,FD,e-mail,etc) を用いて他のユーザに渡す必要がある[7]。

void receive(byte[] byteArray)
他人から受け取ったキューを自分のキューに反映させる。

void redo()
自分の保持しているキューが更新されときに、自分宛てのメッセージが存在したらそれを処理する。receive(byte[]) によりキューが更新されたら必ず行う。
前述のように引数は 参照を保持ではなくて、コピーの保持をする。

A.2 stub

基本的なメソッド起動方法 Object e(String methodName, Object[] argv, int toUser, ReturnListener returnListener) はシームレスではないので Stub compiler を作成した。
Stub を用いると、繁雑さを隠蔽できる。プリミティブ型の “ラッパーか否か” の問題も解決できる。

B Sample Code

下記の SmplAorbSrv クラスをサーバオブジェクトのクラスとする。

```
public class SmplaorbSrv{
    String text = "";
    public String meth(String a, String b){
        text += a;
        return a+b;
    }
}
```

通常のコードは以下のようになる。

```
SmplaorbSrv obj;
String s = obj.meth("Hello,", "world!");
```

Aorb を使用するコードの例は以下のようになる。

```
SmplAorbSrv sas = new SmplAorbSrv();
// サーバインスタンスを作成
aorb.Aorb aorb = new aorb.Aorb(sas, 5, 1);
// サーバインスタンスの公開

SmplAorbReturnListener sarl =
  new SmplAorbReturnListener();
// 戻り値獲得時の処理を作成

Object[] args[] = {"Hello,", "world!"};
aorb.e("meth", args, 2, sarl);
// 呼び出し .

byte[] mylog = aorb.send();
// 送信ログの作成

{log を何らかの通信手段により転送}

byte[] hislog = {log を何らかの通信手段により獲得}

aorb.receive( hislog );
aorb.redo();
// ログの受信 これにより、
// 自分宛のメソッドを起動や戻り値処理が起きる
```

[7] 前述のようにその通信機能は備えていない