# 遠隔操作機能を提供する P2P アプリケーション開発支援システムの実装

梅　津　高　朗　　大阪大学大学院情報科学研究科助教

## 1　まえがき

近年，アプリケーション層マルチキャストに代表される効率的かつ現実的なマルチキャスト配信システムが商用サービスとして利用されるなど，P2P アプリケーションに関する研究・開発が非常に活発に行われている．また，PlanetLab などの P2P アプリケーションを実装して実験を行うためのプラットフォームも一般的になりつつあり，研究分野においても P2P アプリケーション実装に対する要求は高まってきている．しかし，P2P 環境を前提としたアプリケーションの開発は従来型のネットワークアプリケーションに比べて，開発，テストに要する労力が非常に膨大になり，開発現場においても大きな問題となっている．そこで，本研究では P2P アプリケーション用統合開発環境を提案する．

各クライアントがサーバに接続して動作を行うという従来型モデルでは，ネットワークプロトコルを考える際にそれぞれの接続を独立して扱うことができ，設計者は該当する 1 対 1 通信に注視すればよかった．しかし，P2P アプリケーションでは，多数のノード間の相互接続関係を把握した上で作業を進める必要がある．

一般に現在考案されているミドルウェアなどでは図 1 のようにアプリケーションに対して高機能なネットワークインタフェースを提供するような形が多く，ノードを越えたデバッグに対する支援機能はあまり用意されていない．一方，開発手法としては，分散環境を対象とした高信頼性アプリケーション設計環境の研究[1)]や，ノード間をまたぐログ記録方式の研究[2)]など，様々な研究がなされているが，デバッグ等を直接支援するソフトウェアとしての実装は未だ少ない．また，クライアント・サーバモデル用のデバッガ[3)]や，分散実行時のイベント実行系列の可視化を行うツール[4)]など等も提案されているが，ノードの接続が頻繁に変更されるような P2P アプリケーションに特化したものはあまり知られていない．また，実際に P2P アプリケーションを開発する際にはそのアプリケーション専用のシミュレータや開発ツールを用意して実験を行うような作業もしばしば行われている[5)]．

そこで，本研究では一般的な P2P アプリケーションの研究，開発に利用可能な統合開発環境の提案を行う．提案環境におけるミドルウェアは図 2 のように分散デバッガから開発対象となるクライアントソフトウェアを一元的に遠隔操作する機能を有し，開発者はデバッガ上の操作のみにより実験環境に含まれる全てのノードを一括して操作できる．ここでは，研究分野で比較的よく利用されている Java 言語によるアプリケーションを対象に開発環境の実装を行った．

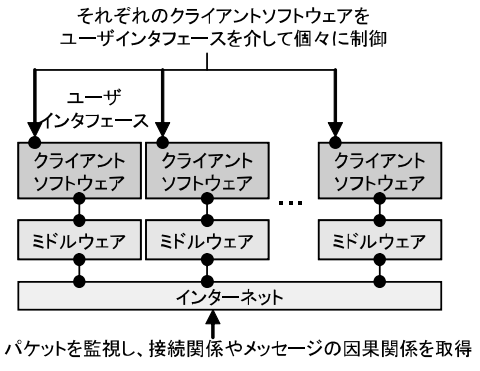


図 1 従来の通信ミドルウェアの概要

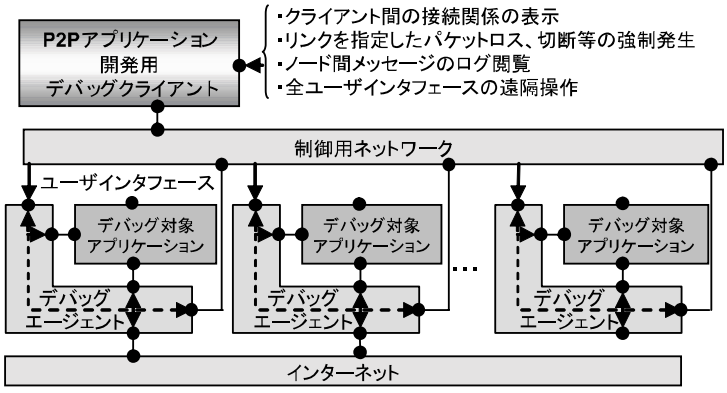


図 2 提案するミドルウェアおよび開発環境の概要

## 2 P2P アプリケーション開発における問題点と提案開発環境

P2P アプリケーション開発においては，従来のサーバ，クライアント型のアプリケーションとは異なる以下のような問題が生じる．

(1) それぞれのノードが多数のノードとの間で協調動作を行うため，複数ノードから非同期に到着するメッセージを処理する必要がある．
(2) 動作検証や性能計測などの際には多数のノードを同時に操作する必要がある．
(3) 特定のサーバを仮定せず，切断，再接続の頻度が高いため，ノード間の接続関係が逐次的に変化し続け，接続関係の把握が困難である．
(4) ネットワーク上に分散配置されたノード群が相互にメッセージを交換するため，全体を通したログの取得が繁雑でデバッグに必要なメッセージの因果関係把握が困難である．

こういった問題を解決するため提案開発環境は，(1)プロトコルの動作を状態機械としてモデル化し，受信メッセージに対する処理を列挙することで簡潔に実装が行えるフレームワーク，(2) ミドルウェアにユーザインタフェースの機能も統合し，デバッガから遠隔操作機能やパケットロス，切断などの擬似的な発生を行う機能(図 5)，(3) 提案エージェントによりリアルタイムに接続関係を調査し，統合デバッガ上でノード間の接続関係がグラフィカルに観察できる機能(図 6)，(4) ノード間で交換される全てのメッセージのログを時系列で統合し，因果関係の調査を支援する機能(図 7，8)，を提供する．実験に利用する多数のノードを遠隔操作するため，それらのノード上であらかじめデバッグエージェントを実行しておき，開発者の端末で動作させるデバッグエージェントからエージェント群を一括操作することで前述の機能が利用できる形を取る(図 2)．

デバッグクライアントからはデバッグエージェントを介して，表 2 に示したクラス群を用いてデバッグ対象アプリケーションを遠隔操作できる．以下，各機能に関して説明する．

**表 1 状態機械によるプロトコルのモデル化例(分散ツリー制御プロトコル)**

| 遷移元状態名 | 受信可能なメッセージ名 | 遷移先状態名(送信されるメッセージ名) |
|---|---|---|
| WaitForAdvertiseRequest | AdvertiseRequest | FinalState() |
| 〃 | 〃 | WaitForRootChangeReplyForParticipate (RootChangeRequest) |
| WaitForRootChangeReplyForParticipate | RootChangeAccept | WaitForParticipateReply (ParticipateRequest) |
| 〃 | RootChangeReject | FinalState() |
| WaitForParticipateReply | ParticipateAccept | FinalState(RootChangeExecute) |
| 〃 | ParticipateReject | FinalState(RootChangeCancel) |
| WaitForParticipateRequest | ParticipateRequest | WaitForRootChangeReplyForAdvertise (RootChangeRequest) |
| 〃 | 〃 | FinalState(ParticipateReject) |
| WaitForRootChangeReplyForAdvertise | RootChangeAccept | WaitForRootChangeAcknowledgement (RootChangeExecute) |

### 2-1 プロトコルの動作の状態機械によるモデル化

一般に，P2P プロトコルは拡張有限状態機械によりモデル化されることが多い．例えば，表 1 にメッセージ配送ツリーを分散的に構築・制御するプロトコルの例を示す．表では，遷移元状態名とその状態で受信可能なメッセージ名，および，そのメッセージを受信した場合に遷移しうる遷移先状態名と遷移の際に送信されるメッセージ名を列挙した．この例では，既存のツリーに新規ノードが参加するような場合のみならず，複数のツリー同士が一つのツリーへと結合し，元のツリーのルートノードの権限を新たなツリー全体のルートノードへと委譲するような動作も規定している．こういった状態遷移表でモデル化を行ったプロトコルを何らかの言語を用いて実装した場合，受信メッセージの種類の判定文など，同様の記述が何度も必要になるためプログラムが非常に繁雑になり，作業ミスによるバグの混入が頻繁に発生する．また，研究段階のプロトコルにおいては，プロトコルの仕様自体もしばしば変更される．例えば，ルートノード権限の委譲を結合前に行った場合と，結合後に行った場合の動作の比較など，試行錯誤を繰り返しながら最適なプロトコルデザインを模索するような場合が多く，その都度，複雑なソースコードの変更が必要となってしまう．

そこで，提案環境では状態機械による記述から Java プログラムスケルトンを生成する機能を提供する．実装するプロトコルを非決定性の状態機械としてモデル化し，表 1 のような形式で，テキストファイルに記述する．提案環境のスケルトン生成プログラムにこのテキストファイルを与えることで，必要なメッセージクラスの宣言や，状態の宣言，受信メッセージの種類による条件分岐と各動作におけるメッセージ送信動作までを備えたスケルトンが出力される．非決定性のある分岐にはプロトコル設計者がこのスケルトンに対して Java 言語で分岐条件を追加して動作を厳密に定めることもできる．なお，以下に述べる提案開発環境は，このスケルトン生成機能により作成したプログラムに限らず，一般のソケット通信プログラムの開発支援ができるよう実装した．ただし，このスケルトン生成機能を利用した場合には各ノードの現在の状態名なども監視することができ，より効率的にデバッグ作業が行える．


図 3 P2P チャットアプリケーション

### 2-2 多数端末の遠隔制御

次に，遠隔の端末を用いて実環境における実験・性能評価を行うためには，多数の端末へとリモートログインを行い，対象とするアプリケーションを実行する必要がある．しかし，開発中には頻繁にバイナリの変更，配布，アプリケーションの起動が必要となるため，遠隔ノードが増加すればするほどその手間は増加する．その手間を軽減するため，提案開発環境ではアプリケーションを動作させる端末群にデバッグエージェントをあらかじめ配置，実行させておく．それらのエージェントに対してデバッグクライアントからデバッグ対象アプリケーションを配布し，エージェント上で実行させる形を取ることで起動・終了や起動時のパラメータ指定などの操作を一括して行えるようにし，開発の手間を軽減する(図 2)．

また，CUI のアプリケーションであれば，GXP[6]などのグリッド/クラスタ用ターミナルソフトを用いることで多数のアプリケーションを一括して遠隔操作することも比較的容易に可能であり，また，シェルスクリプトなどを利用することで自動実行させることも可能である．しかし，近年のアプリケーションは GUI を持つことが基本とされつつあり，特にマルチメディアデータを扱うようなアプリケーションに関しては GUI をユーザに提供することは必須ともいえる．一般に GUI においては，各種のボタンやテキスト入力枠などの他に，実際の操作には必須ではない，ラベルなどのユーザの操作を補助するためのガイド情報が付与され，また，それらの要素はデザイン上の要求から整然と並べられる．

例えば，図 3 に簡単な P2P チャットアプリケーションの例を示す．このアプリケーションは，(a) 参加者の発言を表示するメッセージ表示領域，(b) 参加者の一覧を表示する領域，(c) メッセージを入力するテキスト入力領域，(d) 入力したメッセージを投稿するボタン，の 4 つの主要な要素の他に，それぞれが何を意味するのかをユーザに示すためのラベル類が付与されている．

こういったアプリケーションをそのままの形で遠隔操作することは，図 4 のように画面上の無駄が多いため操作も繁雑となり，あまり効率的とは言えない．一般に，アプリケーション開発者による開発・デバッグ時にはそれらのガイド情報やデザイン上の要求は必ずしも重要ではないため，GUI から操作に必要な種類の

部品のみを自動的に抽出し，集約することで遠隔操作を円滑に行うことを支援する．この例では，最低限の GUI 部品として，(c) テキスト入力領域，(d) 投稿ボタンが操作に必要であるため，図 5 のようにそれらの要素のみを各アプリケーションから抽出し，各ノードでのデバッグ対象アプリケーションの起動終了，標準入出力表示などの機能を加えて並べた形のデバッグ用コンソールを提供する．デバッグ用コンソールに表示された各 GUI 部品は，デバッグ対象アプリケーションの対応する GUI 部品と，デバッグエージェントを介して結びつけられており，いずれか一方に対して行った操作はそのままの形でもう一方でも実行される．この例では，デバッグ用コンソールのテキスト入力領域に対して文字列を入力して投稿ボタンを押すことで，対応するユーザの発言を擬似的に再現できる(図 3，図 5)．

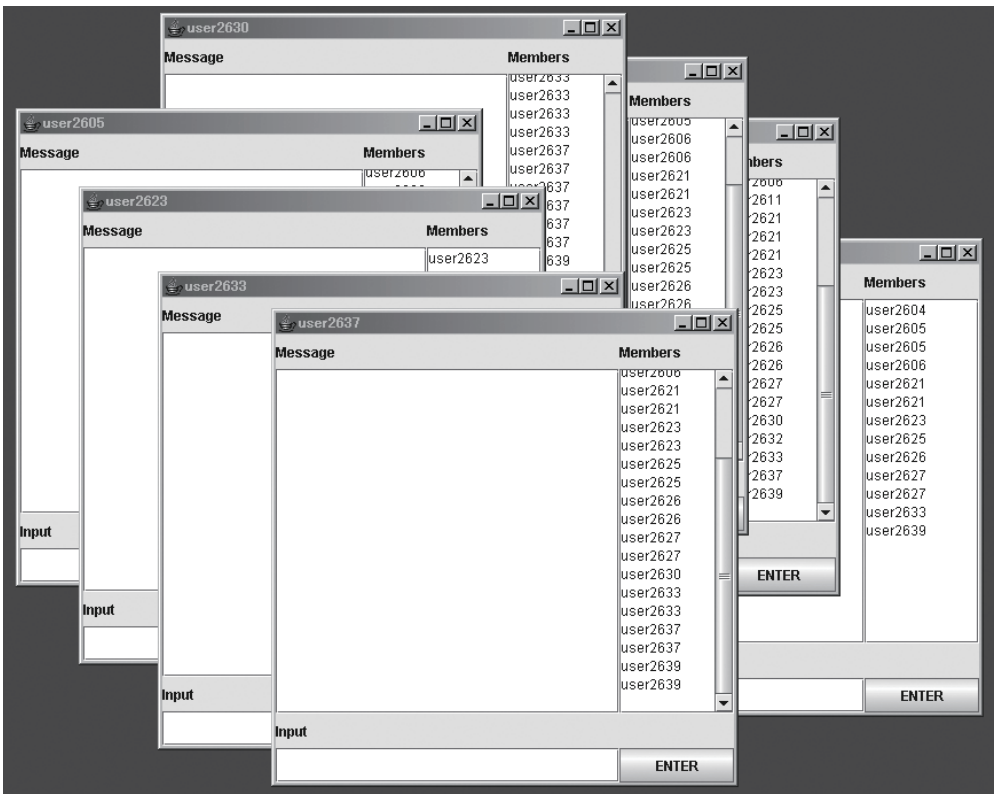


**図 4 GUI を持つアプリケーションの遠隔操作**

### 2-3 端末間の接続関係の表示

P2P アプリケーションにおいては，一般に各端末は他の端末との間でその時々の状況に応じた接続関係を構築する．開発者は，その上でプロトコルの動作や性能を評価することとなる．例えば，効率の良いマルチキャストツリーの分散制御プロトコルを開発する場合を考える．こういったプロトコルの性能を計測する場合，マルチキャストツリーの根ノードといったプロトコル上重要な役割を持つ特定のノードが離脱した場合の挙動の観察や性能の計測はしばしば行われる．しかし，どのノードが木の要になるかはノードの接続順序に依存するため，多数の端末上でアプリケーションを起動して互いに接続させて構築されたノード間の接続関係からそれらを特定することは繁雑な作業となる．そこで，提案開発環境ではノード間の接続関係をグラフィカルに表示する機能を持たせた(図 6)．デバッグ用ライブラリの ServerSocket クラスおよび Socket クラスで標準のソケットライブラリを置き換えることで，ノード間の接続，切断がデバッグエージェントにより監視されデバッグクライアントに通知される．アプリケーション開発者は，図示された接続関係を見ながら接続数の多いノードを遠隔から停止させ，その場合の挙動を調べる，といった作業などが可能となる．

### 2-4 メッセージの因果関係の表示

P2P アプリケーションでは多数のノードが協調動作するため，各ノードでメッセージのログなどを記録できても，それらの関係をリアルタイムに把握することは難しい．そこで，提案開発環境では，デバッグ対象アプリケーションを動作させているノード間でやりとりされるすべてのメッセージをデバッグクライアント上で一元的に閲覧する機能を提供する(図 7，8)．

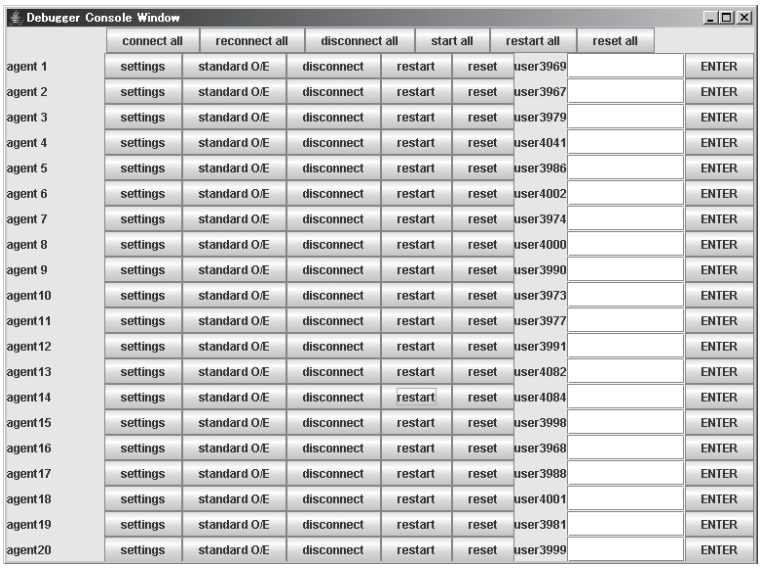


図 5 デバッグクライアント ： デバッグ用コンソール

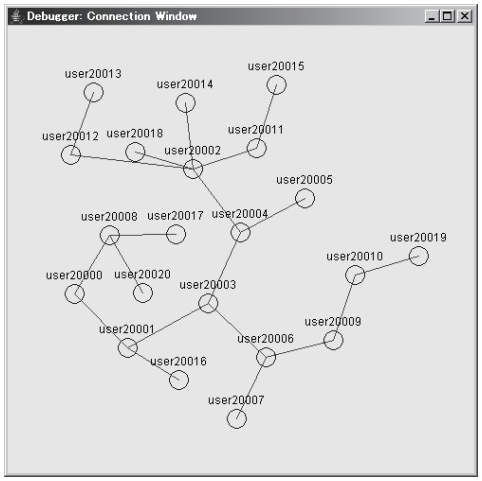


図 6 デバッグクライアント ：
ノード間接続関係表示ウィンドウ

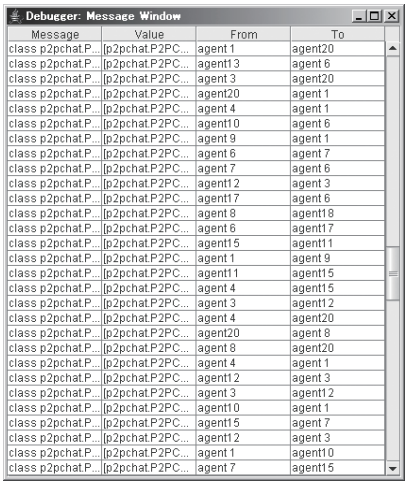


図 7 バッグクライアント ：
メッセージ表示ウィンドウ

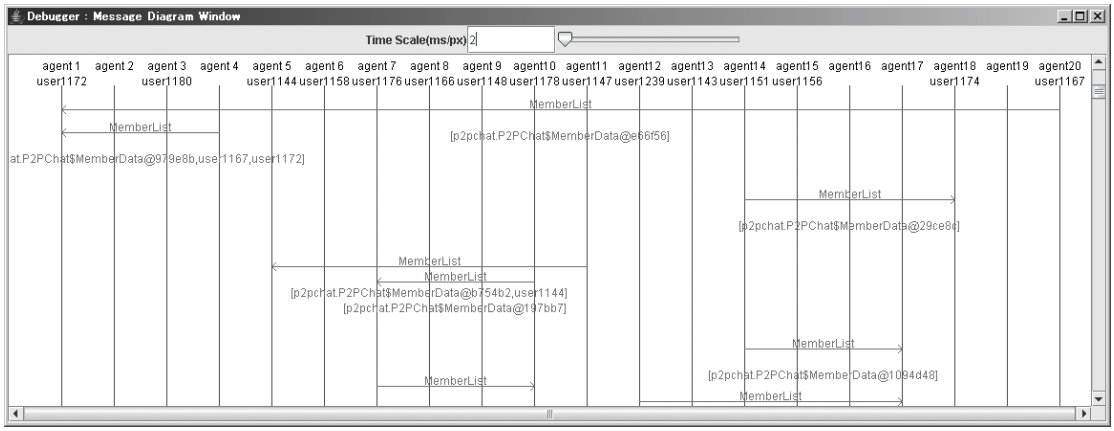


図 8 デバッグクライアント ： 送受信メッセージダイアグラム表示ウィンドウ

ここで，図 7 のメッセージ表示は，単純な時系列に従った表示のみならず，メッセージの種類や送受信端末などからなる条件を指定してメッセージの抽出を行い表示する機能も有する．また，図 8 のメッセージダイアグラムでは，GUI 操作により表示するタイムスケールを変更することで特定の瞬間のメッセージのやりとりなどを拡大表示することもでき，メッセージの因果関係追跡を支援する．

## 3　提案開発環境の実装

以下では，提案開発環境の実装について述べる．

### 3-1 デバッグエージェント/クライアントの動作

デバッグエージェントはサーバとして動作する．まず，あらかじめデバッグに使用する端末群全てでデバッグエージェントを動作させておく．デバッグクライアントは，起動時にあらかじめ指定されたアドレスへとパケットを送信することで利用可能なデバッグエージェントを検索して接続する(図 2)．その後，デバッグクライアントは起動時に指定されたアプリケーションをデバッグクライアントへと配布して実行させる．構築した多数の端末を含む環境は複数のアプリケーション開発に使い回して利用することができるため，あらかじめ大規模なデバッグ用環境を用意しておくことで開発の手間を大幅に削減できる．

表 2　デバッグ用ライブラリの主なクラスとその機能

| デバッグ用クラス | 対応する Java 標準クラス | 機能 |
|---|---|---|
| ServerSocket | java.net.ServerSocket | TCP ソケット待ち受け端点．接続状態を監視してデバッグクライアントに通知 |
| Socket | java.net.Socket | TCP ソケット端点．送受信されるメッセージを監視してデバッグクライアントに通知 |
| DatagramSocket | java.net.DatagramSocket | UDP ソケット端点．送受信されるメッセージを監視してデバッグクライアントに通知 |
| JFrame | javax.swing.JFrame | GUI ウィンドウ部品．ウィンドウタイトルをデバッグクライアントに通知 |
| JButton | javax.swing.JButton | GUI ボタン部品．ボタンの押下動作を遠隔操作 |
| JTextField | javax.swing.JTextField | GUI テキスト入力部品．テキスト入力を遠隔操作 |

### 3-2 デバッグエージェント/クライアントの動作

アプリケーションの遠隔操作デバッグ対象アプリケーションを遠隔操作するため，デバッグクライアントはアプリケーションのバイトコードファイルを読み込む際に，表 2 に示すようなクラスが利用されている箇所を検索し，該当部分をデバッグ用ライブラリが呼び出されるように書き換える．書き換えられる部分は，Java 標準のクラス群の内，通信や GUI 制御に関わるクラス群であり，デバッグ用ライブラリのこれらのクラスは，標準クラスの持つ機能に加えて，通信や GUI 操作を監視してデバッグクライアントへと通知する機能や，逆にデバッグクライアントからデバッグ対象アプリケーションを遠隔操作する機能を備えている．ここでは，Java バイトコードの情報の表示やその編集を行うために開発された Javassist[7]を用いて，バイトコードの変更を実装した．

以下では，

```
button = new javax.swing.JButton("実行");
```

のように，Java 標準のボタンオブジェクトを利用する動作を例にとって説明を行う．この場合，ロード時に該当部分のバイトコードが改変され，

```
button = new debugger.JButton("実行");
```

のように，デバッグ用ライブラリのボタンオブジェクトを作成するような動作に書き換えられる．この

debugger.JButton クラスは，通常のボタンの動作に加えて以下の機能を持つ．

(1) ボタンをウィンドウ内に表示する際に，それをデバッグクライアント側に通知し対応するリモートボタンを表示させる．
(2) ボタンがクリックされた際に，それをデバッグクライアントに通知する．
(3) デバッグクライアントは，対応するリモートボタンがクリックされた際にデバッグエージェントにそれを通知し，アプリケーションにボタンがクリックされた場合の動作を行わせる．

他の GUI 部品に関しても同様に遠隔操作機能を持ち，遠隔で起動したアプリケーションをデバッグクライアント側から操作することが可能となる．また，デバッグ用ライブラリの debug.Socket クラスは，Java の Socket クラスの動作に加えて，そのソケットを通して行われた通信を監視し，デバッグクライアントに通知する機能を持つ．

## 4 性能評価

提案開発環境の実用性を評価するため，オーバーヘッドに関していくつかの実装実験を行った．ここで開発対象アプリケーションは，図 3 の P2P 通信に基づくチャットプログラムとした．

### 4-1 開発対象アプリケーションの配布・実行時間

まず始めに，デバッグクライアントとデバッグエージェントを接続した状態から，デバッグ対象アプリケーションのバイナリをエージェントへと転送し，エージェントで対象アプリケーションを起動，デバッグ作業が可能になるまでの時間を計測した．実験には(a)Pentium2.4GHz，メモリ 1GB の Windows XP をインストールした PC と(b)Xeon2.0GHz，メモリ 1GB の Windows 2003 Server をインストールした PC を用い，Java の環境としては JDK1.5 を用いた．デバッグクライアントとエージェントを，(1) 同一マシン上(PC(a) 上で実施)，(2) 同一 LAN 内(PC(a)，PC(b)を 1000Base-T で接続して実験)，(3) 遠隔地(PC(b)に対して PC(a) を CATV 回線を介した PPTP プロトコルによる VPN により接続して実験)，(4) 遠隔地(同・逆方向に接続) に一対一に配置し接続を行った．CATV 回線は,ベストエフォートで最高性能は登り 8Mbps，下り 2Mbps であり，実験中の実効値はその 8 割程度で安定していた．その後，クライアント側を操作して，エージェント側で 20 インスタンスのデバッグ対象アプリケーションを順次起動させ，1 インスタンスあたりの起動に要した時間を計測した結果を図 9 に示す．また，グラフにはそれぞれのマシン間でファイル転送を行った場合の転送速度の実測値も示した．

回線が細い場合(32KB/s)，1 回のデバッグ対象アプリケーション起動に 5 秒程度の時間を要してしまっている．これは，アプリケーション起動毎に，そのバイナリの全て(約 26KB) を転送してから起動しているためと思われる．一般にデバッグ作業においては，デバッグ対象アプリケーションに多少の変更を加えながら何度も実行することになる．そのため，全てのバイナリを転送して初期化するのではなく，変更のあったクラスのみ，もしくはバイトコードバイナリの差分のみを転送し，また，文献 8)のような方法を応用し，更新が行われなかったクラスに関しては以前に初期化したものを再利用するような方法での高速化が可能であると思われる．

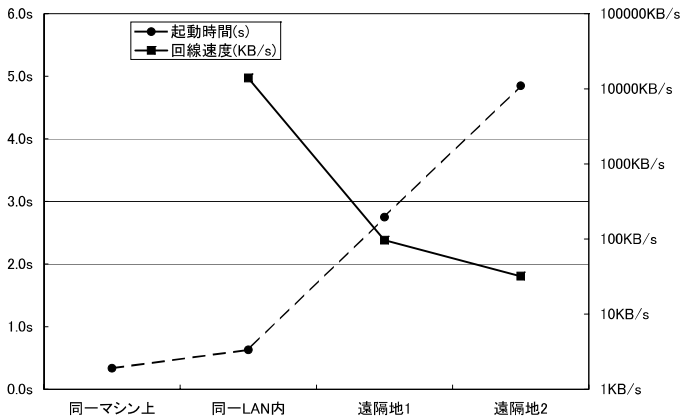


**図 9 デバッグ対象アプリケーション起動に要する時間**

### 4-2 遠隔操作に伴う通信オーバーヘッド

次に，遠隔操作を行った場合の通信オーバーヘッドを計測した．操作に伴う遅延は一般的なリモート操作ツールに対して遜色のない程度であり，数十エージェント程度であれば実用に足ると思われる．提案開発環境では，デバッグ対象アプリケーションでイベント(パケットの送受信等) が発生した場合や，操作を行った場合にデバッグクライアントとエージェントの間で TCP により通信を行い,遠隔操作･監視を実現している．ここでは，いくつかのイベントについて，その際に発生している通信オーバーヘッドを計測した．それぞれの操作を行った場合にやりとりされるパケット量を計測した結果を表 3 に示す．表では，各イベントや操作の内容に関わるデータ量(通信イベントにおけるアプリケーション間で送受信されたパケットのデータ量や，テキスト入力イベントにおける入力されたテキスト自体のデータ量) をのぞいた値を示した．それぞれ，Java 標準の直列化を用いているため，オーバーヘッドがやや大きくなっている．通信関係のイベントでは，ソケット端点を表すオブジェクトが多くの情報を保持しているため，特に大きなオーバーヘッドを伴っている．

また，現在の実装では，アプリケーション間で送受信されたデータそのものもエージェントからクライアントへと転送してから解析処理を行って可視化(図 7,8) しているため，マルチメディアを扱うようなアプリケーションではエージェント・クライアント間の通信量の増加が問題となる．文献 9) の実行系列を圧縮して保存するような手法などを参考に，エージェント側で要約してからクライアントへと通知する拡張を今後の課題としたい．

**表 3 遠隔操作に伴うオーバーヘッド**

| イベント・操作 | 通信オーバーヘッド |
|---|---|
| アプリケーション間の TCP 接続イベント | 約 300Bytes |
| アプリケーション間の TCP 切断イベント | 約 300Bytes |
| アプリケーション間の通信イベント | 約 700Bytes |
| ボタンクリック操作 | 約 70Bytes |
| テキストメッセージの入力操作 | 約 100Bytes |

## 4 まとめと今後の課題

本稿では P2P アプリケーションの開発，性能評価を支援するための統合開発環境を提案した．提案開発環境では，あらかじめ複数の遠隔ノードで遠隔操作用のデバッグエージェントを動作させておき，それらを開発者の利用する端末で動作させるデバッグクライアントから一括して遠隔操作することで，デバッグや動作実験を行う手間を軽減する．

今後の課題としては，以下のような問題に対する解決を考えている．まず，通信量の削減などをあまり考慮せず実装を行っているため，操作に伴うオーバーヘッドが比較的大きくなってしまっている．現状でも数十程度のインスタンスを遠隔操作する場合には十分な性能であることは確認できたが，数千規模のノード数でも利用できるよう拡張を行いたい．通信量を削減するよう改良するほか，デバッグエージェントを階層的に接続するなどして，よりスケーラビリティを持たせたアーキテクチャを模索していく．次に，現在は単体動作するデバッグ用クライアントとして試作を進めているが，これを Eclipse[10]などの統合開発環境に組み込むことでアプリケーション全体のデバッグを統合的に行えるよう環境を整備することを考えている．

また，P2P プロトコルやアプリケーション実装実験などに実際に利用し，必要とされる機能を順次追加実装していく．セキュリティー機能を持たせ，LAN 内のみならず PlanetLab[11]などのオープンな遠隔環境でも利用できるように環境を整える．最終的にはオープンソースのソフトウェアとして公開し，広く利用されるようになることを目指す．

## 【参考文献】

1) E. Martel, F. Guerra, J. Miranda: “EPDAModeller: A Tool for Modelling a Distributed Programming Environment”, Proc. of 7th Int. Symp. on Comp. and Comm. (ISCC’02), pp. 245.250 (2002)

2) V.K. Garg, B. Waldecker: “Detection of Weak Unstable Predicates in Distributed Programs”, IEEE Transactions on Parallel and Distributed Systems, Vol. 5, No. 3, pp. 299-307 (1994)
3) The IBM Distributed Debugger: advanced local and client/server solution, http://www-306.ibm.com/ software/awdtools/debugger/
4) T. Kunz, J. P. Black, D. J. Taylor and T. Basten: “Poet: Target-System Independent Visualizations of Complex Distributed-Application Executions”, The Computer Journal (special issue on building parallel and distributed systems), Vol. 40, No. 8, pp. 499.512 (1997)
5) 金子勇, アスキー書籍編集部(編)：“Winny の 技術”, アスキー(2005). 8 情報処理学会論文誌 1959
6) GXP：Grid/Cluster Shell, http://www.logos.ic.i.utokyo. ac.jp/phoenix/ gxp quick man ja.shtml
7) C. Shigeru and N. Muga: “An Easy-to-Use Toolkit for Efficient Java Bytecode Translators”, Proc. of 2nd Int. Conf. on Generative Programming and Component Engineering (GPCE ’03), LNCS 2830, pp.364-376, Springer-Verlag (2003)
8) W. Bernard, C. Grzegorz, D. Laurent：“Dynamically Loaded Classes as Shared Libraries: An Approach to Improving Virtual Machine Scalability”, Proc. of Int. Parallel and Distributed Processing Symp. (IPDPS’03), pp. 38b
9) W. Tao, R. Abhik：“Using Compressed Bytecode Traces for Slicing Java Programs”, Proc. of 26th Int. Conf. on Software Engineering (ICSE’04), pp. 512-521
10) Eclipse: an universal tool platform - an open extensible IDE, http://www.eclipse.org/
11) PlanetLab: An open platform for developing, deploying and accessing planetary-scale services, http://www.planet-lab.org/

## 〈発 表 資 料〉

| 題　　名 | 掲載誌・学会名等 | 発表年月 |
|---|---|---|
| P2P アプリケーションの開発と性能評価のための統合開発環境の提案 | 情報処理学会論文誌, Vol. 47, No. 7 | 2006 年 7 月 |
| アプリケーション層マルチキャストミドルウェアの実装と PlanetLab 上での評価 | 情報処理学会 第 128 回「マルチメディア通信と分散処理」研究会, Vol. 2006 No. 96 | 2006 年 9 月 |
| A Middleware for Implementation and Evaluation of Application Layer Multicast Protocols in Real Environments | 17th International workshop on Network and Operating Systems Support for Digital Audio & Video (NOSSDAV 2007) | 2007 年 6 月 |
| アプリケーション層マルチキャストプロトコルの設計開発および性能評価を支援するミドルウェアの設計と実装 | 情報処理学会 マルチメディア，分散，協調とモバイル(DICOMO 2007)シンポジウム論文集 | 2007 年 7 月 (to appear) |