

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

梅津 高朗[†] 池田 直徒[†] 東野 輝夫[†]

近年、従来のクライアント、サーバモデルのアプリケーションに代わるものとして P2P アプリケーションの研究がさかんに行われている。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。提案開発環境においては、デバッグ・実装実験を行う端末群であらかじめデバッグを支援するデバッグエージェントを実行しておく。デバッグクライアントから多数のエージェントを一括して遠隔操作し、提案環境の提供するデバッグ対象のアプリケーションの配布や開始・終了、GUI の一元的な操作などの機能を開発者に対して提供することで P2P アプリケーションの開発を支援する。

An Unified Development Tool for Design and Evaluation of P2P Applications

TAKA AKI UME DU,[†] NAOTO IKEDA[†] and TERUO HIGASHINO[†]

Recently P2P applications are considered to be alternative techniques of usual client-server style applications and many studies about them have been carried on. However, to develop P2P applications is much more complicated work than that of usual client-server style applications since it is very difficult to implement such applications correctly and test their correctness. So such difficulty has become a great problem in research activities. In this paper, we propose a unified development tool for designing P2P applications. Our development tool is constructed with debug agents and a debug client. Debug agents are executed on the terminals for debug and evaluation in advance, and they support the debugging work. Developers can control those debug agents all together through the debug client.

1. ま え が き

近年、従来のクライアント、サーバモデルのアプリケーションに代わるものとして P2P アプリケーションの研究がさかんに行われている。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うえ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、デバッグなどを直接支援するソフトウェアとしての実装はまだまだ少ない。また、クライアント・サーバモデル用のデバッグ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

[†] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

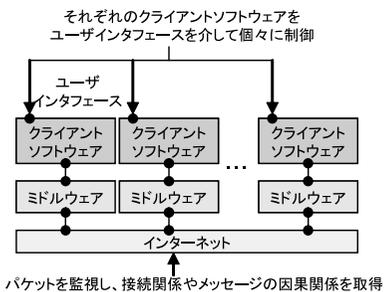


図 1 従来の通信ミドルウェアの概要図

Fig. 1 Usual communication middleware.

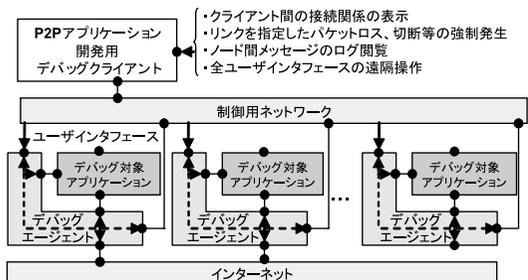


図 2 提案するミドルウェアおよび開発環境の概要図

Fig. 2 Proposing middleware and development tool.

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すべてのノードを一括して操作できる。ここでは、研究分野で比較的良好に利用されている 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。

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複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すべ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, 図 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とる (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。

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。たとえば、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だけでなく、複数のツリーどうしが 1 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や判定文など、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複雑になり、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。たとえば、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、そのつど、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状態機械としてモデル

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)
 Table 1 An EFSM like example specification of a distributed tree control protocol.

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)
"	RootChangeReject	FinalState(ParticipateReject)
WaitForRootChangeAcknowledgement	RootChangeAcknowledgement	FinalState(ParticipateAccept)

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.

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, 開発中には頻りにバイナリの変更, 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とることで起動・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, GXP⁶⁾

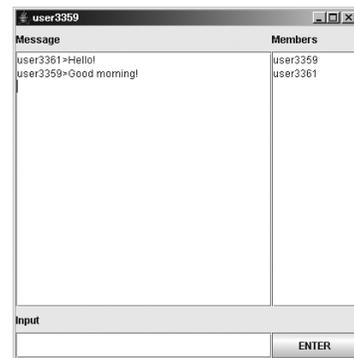


図 3 P2P チャットアプリケーション
 Fig. 3 A P2P chat application.

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ほかに, 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.

たとえば, 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

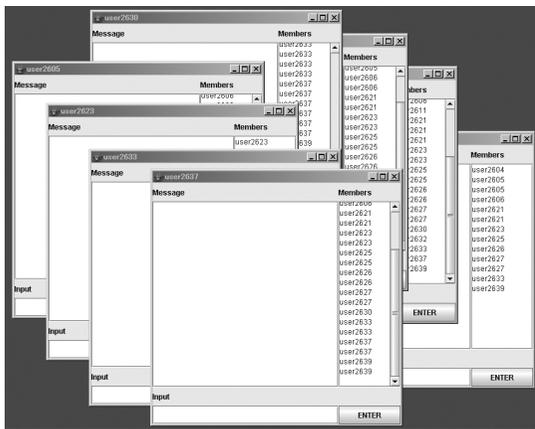


図 4 GUI を持つアプリケーションの遠隔操作

Fig. 4 Remote control of a GUI based application.

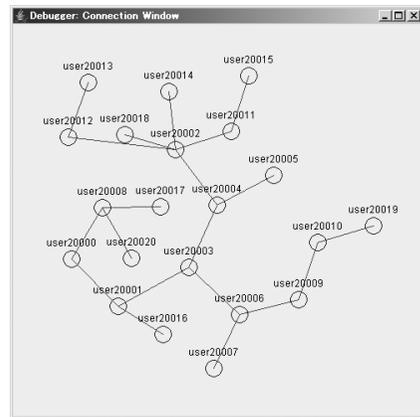


図 6 デバッグクライアント：ノード間接続関係表示ウィンドウ

Fig. 6 Debug client: connection graph window.

	connect all	reconnect all	disconnect all	start all	restart all	reset all	
agent 1	settings	standard O.E	disconnect	restart	reset	user3969	ENTER
agent 2	settings	standard O.E	disconnect	restart	reset	user3967	ENTER
agent 3	settings	standard O.E	disconnect	restart	reset	user3979	ENTER
agent 4	settings	standard O.E	disconnect	restart	reset	user4041	ENTER
agent 5	settings	standard O.E	disconnect	restart	reset	user3986	ENTER
agent 6	settings	standard O.E	disconnect	restart	reset	user4002	ENTER
agent 7	settings	standard O.E	disconnect	restart	reset	user3974	ENTER
agent 8	settings	standard O.E	disconnect	restart	reset	user4000	ENTER
agent 9	settings	standard O.E	disconnect	restart	reset	user3990	ENTER
agent 10	settings	standard O.E	disconnect	restart	reset	user3973	ENTER
agent 11	settings	standard O.E	disconnect	restart	reset	user3977	ENTER
agent 12	settings	standard O.E	disconnect	restart	reset	user3991	ENTER
agent 13	settings	standard O.E	disconnect	restart	reset	user4002	ENTER
agent 14	settings	standard O.E	disconnect	restart	reset	user4004	ENTER
agent 15	settings	standard O.E	disconnect	restart	reset	user3998	ENTER
agent 16	settings	standard O.E	disconnect	restart	reset	user3988	ENTER
agent 17	settings	standard O.E	disconnect	restart	reset	user3988	ENTER
agent 18	settings	standard O.E	disconnect	restart	reset	user4001	ENTER
agent 19	settings	standard O.E	disconnect	restart	reset	user3981	ENTER
agent 20	settings	standard O.E	disconnect	restart	reset	user3999	ENTER

図 5 デバッグクライアント：デバッグ用コンソール

Fig. 5 Debug client: main console.

るボタン、の 4 つの主要な要素のほかに、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、あまり効率的とはいえない。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付けられており、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)。

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ときどきの状況に応じた接続関係を構築する。開発者は、そのうえでプロトコルの動作や性能を評価することとなる。たとえば、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。

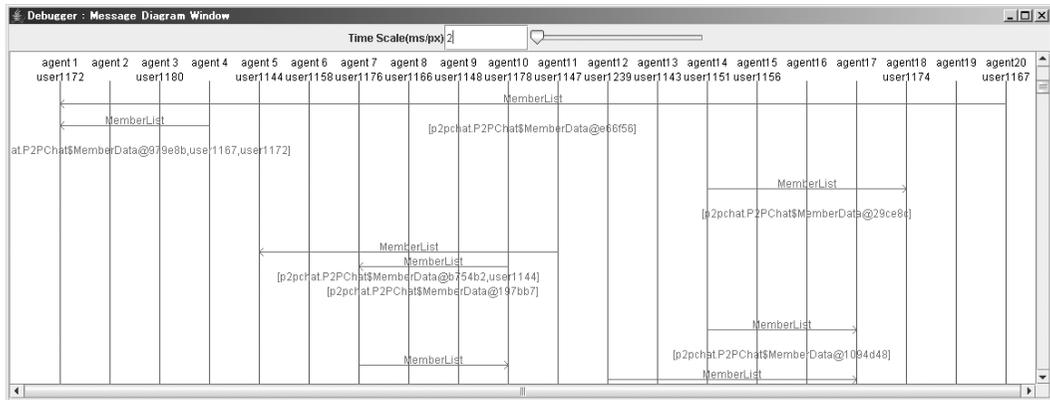


図 8 デバッグクライアント：送受信メッセージダイアグラム表示ウィンドウ

Fig. 8 Debug client: message diagram window.

Message	Value	From	To
class p2pchat.P...	[p2pchat.P2PC...	agent 1	agent20
class p2pchat.P...	[p2pchat.P2PC...	agent 3	agent 6
class p2pchat.P...	[p2pchat.P2PC...	agent 3	agent20
class p2pchat.P...	[p2pchat.P2PC...	agent20	agent 1
class p2pchat.P...	[p2pchat.P2PC...	agent 4	agent 1
class p2pchat.P...	[p2pchat.P2PC...	agent 10	agent 6
class p2pchat.P...	[p2pchat.P2PC...	agent 9	agent 1
class p2pchat.P...	[p2pchat.P2PC...	agent 6	agent 7
class p2pchat.P...	[p2pchat.P2PC...	agent 7	agent 6
class p2pchat.P...	[p2pchat.P2PC...	agent 12	agent 3
class p2pchat.P...	[p2pchat.P2PC...	agent 17	agent 6
class p2pchat.P...	[p2pchat.P2PC...	agent 8	agent18
class p2pchat.P...	[p2pchat.P2PC...	agent 6	agent17
class p2pchat.P...	[p2pchat.P2PC...	agent 5	agent11
class p2pchat.P...	[p2pchat.P2PC...	agent 1	agent 9
class p2pchat.P...	[p2pchat.P2PC...	agent 11	agent15
class p2pchat.P...	[p2pchat.P2PC...	agent 4	agent15
class p2pchat.P...	[p2pchat.P2PC...	agent 3	agent12
class p2pchat.P...	[p2pchat.P2PC...	agent 4	agent20
class p2pchat.P...	[p2pchat.P2PC...	agent20	agent 8
class p2pchat.P...	[p2pchat.P2PC...	agent 8	agent20
class p2pchat.P...	[p2pchat.P2PC...	agent 4	agent 1
class p2pchat.P...	[p2pchat.P2PC...	agent 12	agent 3
class p2pchat.P...	[p2pchat.P2PC...	agent 3	agent12
class p2pchat.P...	[p2pchat.P2PC...	agent 10	agent 1
class p2pchat.P...	[p2pchat.P2PC...	agent 5	agent 7
class p2pchat.P...	[p2pchat.P2PC...	agent 2	agent 3
class p2pchat.P...	[p2pchat.P2PC...	agent 1	agent10
class p2pchat.P...	[p2pchat.P2PC...	agent 7	agent 5

図 7 デバッグクライアント：送受信メッセージ表示ウィンドウ

Fig. 7 Debug client: message observation window.

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, 図8)。

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だけでなく、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すべ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。

3.2 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標準のクラス群のうち、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

表 2 デバッグ用ライブラリの主なクラスとその機能

Table 2 Main classes and their functions of debug library.

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

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 Javassist⁷⁾ 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

まず初めに、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

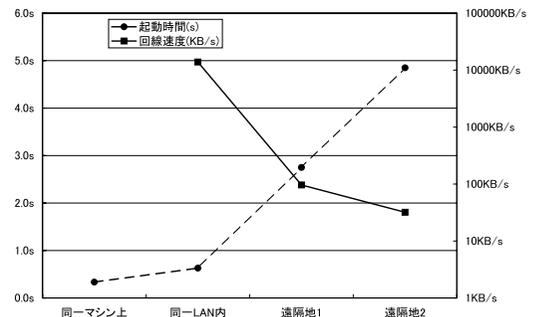


図 9 デバッグ対象アプリケーション起動に要する時間

Fig. 9 Spent time for starting up target application.

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) Pentium2.4 GHz, メモリ 1 GB の Windows XP をインストールした PC と (b) Xeon2.0 GHz, メモリ 1 GB の Windows 2003 Server をインストールした PC を使い、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 1 対 1 に配置し接続を行った。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 (32 KB/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ごとに、そのバイナリのすべて (約 26 KB) を転送してから起動し

ベストエフォート上り 8 Mbps, 下り 2 Mbps. 実効値はその 8 割程度で安定。

表 3 遠隔操作にともなう通信オーバーヘッド

Table 3 Communication overhead with remote control.

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 300 Bytes
アプリケーション間の TCP 切断イベント	約 300 Bytes
アプリケーション間の通信イベント	約 700 Bytes
ボタンクリック操作	約 70 Bytes
テキストメッセージの入力操作	約 100 Bytes

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、すべ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。

4.2 遠隔操作にともな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。操作にともなう遅延は一般的なり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（パケットの送受信など）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持っているため、特に大きなオーバーヘッドをとまっている。

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, 図 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。

5. 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に実装を行っているため、操作にともな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 Eclipse¹⁰⁾ 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。セキュリティ機能を持たせ、LAN 内だけでなく PlanetLab¹¹⁾ 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) Martel, E., Guerra, F. and Miranda, J.: EPDA-Modeller: A Tool for Modelling a Distributed Programming Environment, *Proc. 7th Int. Symp. on Comp. and Comm. (ISCC'02)*, pp.245-250 (2002).
- 2) Garg, V.K. and Waldecker, B.: Detection of Weak Unstable Predicates in Distributed Programs, *IEEE Trans. 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) Kunz, T., Black, J.P., Taylor, D.J. and

Basten, T.: 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).
- 6) GXP: Grid/Cluster Shell.
http://www.logos.ic.i.u-tokyo.ac.jp/phoenix/gxp-quick_man_ja.shtml
- 7) Shigeru, C. and Muga, N.: An Easy-to-Use Toolkit for Efficient Java Bytecode Translators, *Proc. 2nd Int. Conf. on Generative Programming and Component Engineering (GPCE'03)*, LNCS 2830, pp.364-376, Springer-Verlag (2003).
- 8) Bernard, W., Grzegorz, C. and Laurent, D.: Dynamically Loaded Classes as Shared Libraries: An Approach to Improving Virtual Machine Scalability, *Proc. Int. Parallel and Distributed Processing Symp. (IPDPS'03)*, p.38b (2003).
- 9) Tao, W. and Abhik, R.: Using Compressed Bytecode Traces for Slicing Java Programs, *Proc. 26th Int. Conf. on Software Engineering (ICSE'04)*, pp.512-521 (2004).
- 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/>

(平成 17 年 11 月 30 日受付)

(平成 18 年 5 月 9 日採録)



梅津 高朗 (正会員)

平成 13 年大阪大学大学院基礎工学研究科情報数理系専攻博士前期課程修了。平成 14 年同大学大学院博士後期課程退学後, 同大学院情報科学研究科助手。博士 (情報科学)。P2P アプリケーション用ミドルウェアの研究に従事。



池田 直徒

平成 18 年大阪大学基礎工学部情報科学科卒業。現在, 同大学大学院情報科学研究科博士前期課程在学中。長距離広帯域ネットワークの QoS に関する研究に従事。



東野 輝夫 (正会員)

昭和 54 年大阪大学基礎工学部情報工学科卒業。昭和 59 年同大学大学院基礎工学研究科博士課程修了。同年同大学助手。現在, 同大学大学院情報科学研究科教授, 工学博士。分散システム, 通信プロトコル, モバイルコンピューティング等の研究に従事。電子情報通信学会, ACM 各会員。IEEE Senior Member。