分散型ソフトウェアの新しい構成方式の研究
(研究課題番号：16016278)

研究代表者　杉山安洋　日本大学工学部情報工学科

1　研究の概要

インターネットの発達にともない，分散型のソフトウェアの重要性が広く認識され，その需要も爆発的に増大している．しかし，分散型のソフトウェアを開発することは，技術的に難しいことも多い．

分散型のソフトウェアを開発するためには，そのアプリケーション本来の機能であるアプリケーションロジックに加えて，その機能を複数の計算機に分散させて実行するための分散ソフトウェア特有のコードを開発しなければならない．分散処理のためのコードには，複数の計算機に分散した機能をネットワークを経由して連携して実行させるためのネットワーク処理やスレッド制御などが含まれる．このような分散処理のためのコードをアプリケーションロジックと合わせて開発することは，アプリケーションロジックと分散処理の混同を招き，開発作業を必要以上に複雑にしてしまう．また，アプリケーションロジックの開発者が，いつも分散処理に精通しているとは限らない．既に開発済みのアプリケーションを分散化する必要が発生する場合もある．既存ソフトウェアの変更は新規開発よりも工数がかかる場合も多い．

しかし，アプリケーションロジックと分散処理のためのコードとを独立して開発できれば，回避できる問題も多い．アプリケーションロジックと分散処理は，それぞれの分野を得意とする別の開発者が担当し効率よく開発できるようになり，また，既存のスタンドアロンのアプリケーションには外付けで分散処理を追加できるようになる．

本研究の目的は，分散システムの開発にあたって，アプリケーションロジックとネットワーク処理を分離して開発できる方式を確立し，その方式を支援するツールを開発することにある．アプリケーションロジックとネットワーク処理とを独立して開発できれば，アプリケーションロジックとネットワーク処理は，それぞれの分野を得意とする別の開発者が担当し効率よく開発できるようになり，また，既存のアプリケーションには外付けでネットワーク処理を追加できるようになる．即ち，分散システム開発における関心事の分離である．

筆者の研究室では，分散オブジェクト開発支援システム *TRMI(Transparent Remote Method Invocation)* を開発してきた．*TRMI* の初期バージョン [1] では，ネットワークの透過性を実現し，分散オブジェクトをなるべく簡単に開発できることを目的として研究してきた．本研究では，*TRMI* をさらに発展させ，分散オブジェクトを開発する際の関心事の分離ができるようにした．図 1 は，本研究で発展させた *TRMI* のアー

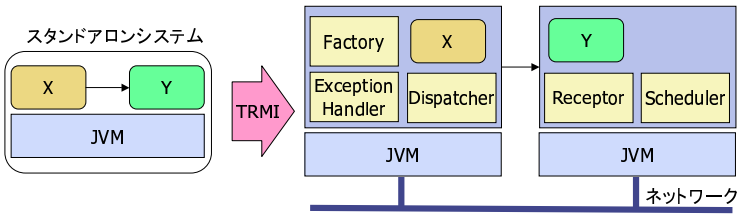


図 1: *TRMI* の概要

キテクチャの概略を示すものである．*TRMI* においては，スタンドアロンのアプリケーションロジックを実装したオブジェクトは，ネットワーク処理やスレッド制御，例外処理，オブジェクト配置などの機能を持つ部品と組み合わせるだけで分散オブジェクトとすることができる．

図 1 で，*dispatcher* と *receptor* はネットワーク処理を担当する部品である．全てのネットワーク処理はこれらの部品内に隠蔽されている．アプリケーションロジックを含む他の部品は，ネットワーク処理に対して一切関知しなくて良い．さらに，各サーバオブジェクトは *exception handler* と *scheduler* および *factory* オブジェクトを持つ．*Exception handler* オブジェクトは，サーバのメソッドをネットワーク化されたマルチスレッド環境で実行する際に発生する例外を処理する機能を持つ．*Scheduler* オブジェクトは，サーバのメソッドを同時に実行する複数のスレッドの同期制御を行う役割を持っている．*Factory* オブジェクトは，分散環境におけるサーバオブジェクトの生成や，遠隔ホストへの配置機能を持つ．

研究は，現在分散システムの開発に広く使用されている Java 言語を対象として行った．また，方法論の研究のみではなく，その方法論を実現するためのツールを合わせて開発し，実用化を目指した研究を行った．また，開発するツールは，分散ソフトウェアを開発する際に使用するだけでなく，実行時における開発者やシステム管理者の負担を軽減する機能を含んだ総合的なツールにすることを狙った．現時点までに，*TRMI* のおおまかな実装は完了し，クライアント・サーバ型，ピア・ツー・ピア型並びにグリッド型の分散システムを *TRMI* を用いて関心事を分離しながら開発できるようになった．

2　研究期間と研究経費の配分額

本研究は，平成 16 年度と 17 年度の 2 年間に渡って進められた．この間に配分された研究経費は下記のとおりである．

年度 (平成)	16	17	合計
研究費 (千円)	4,900	2,900	7,800

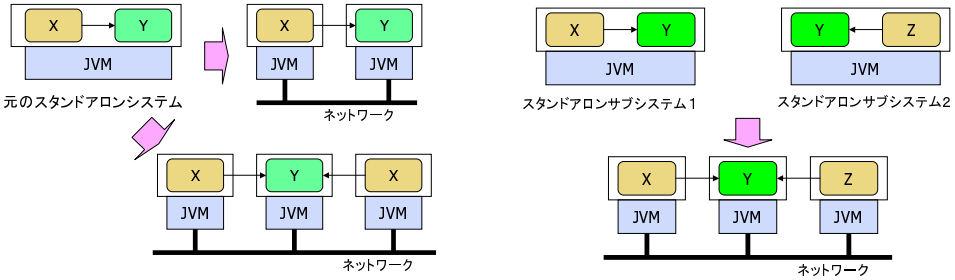
3　研究成果

TRMI は，*Java* 言語で分散システムを開発する際に，アプリケーションロジックと分散処理のためのコードを独立して開発することを可能とするシステムである．*TRMI* を用いると，スタンドアロンのソフトウェアを，それとは別に開発した分散処理部品と組み合わせることにより，分散型のソフトウェアに発展させることができるようになる．

アプリケーションロジックと分散処理を分離して開発する際には，アプリケーションロジックから分散処理が透過的に使用できることが望ましい．アプリケーションロジックから分散処理を意識したコーディングをしなければいけないようであれば，両者を分離したとしても，その効果が半減してしまう．*TRMI* を用いると，アプリケーションロジック中の各々のオブジェクトには分散システムであることを意識したコードを組み込むことなく，分散化させることができる．即ち，クライアントであるオブジェクトは，別の Java 仮想マシン（*JVM*）上のサーバのメソッドを，あたかも同一の *JVM* 上にそのサーバが存在するかのように呼び出すことが可能になる．しかも，クライアントやサーバには，他の分散オブジェクト技術では必要であった，分散処理のための特別のコードを追加する必要がない．

TRMI においてアプリケーションロジックと組み合わせることのできる部品には，図 1 でも示した通り，*dispatcher*，*receptor*，*exception handler*，*scheduler* および *factory* がある．

ネットワーク処理は，*dispatcher* と *receptor* と呼ばれる 2 つの部品により処理される．*Dispatcher* オブジェクトは，クライアント側の部品であり *receptor* オブジェクトはサーバ側の部品である．*Dispatcher* と *receptor* は，ネットワークに関する処理を全て担当する．*Dispatcher* と *receptor* が通信する手順は，これらのオブジェクト間のみで決められており，他のオブジェクトからは隠蔽されている．従って，*dispatcher* と *receptor* をペアで入れ換えることにより，任意の通信プロトコルを使用できるようになる．なお，*TRMI* が標準で使用するプロトコルは Java RMI である．



(1) 分離型　　(2) 統合型

図 2: 基本的な分散化パターン

Exception handler オブジェクトは，サーバのメソッドをネットワーク化されたマルチスレッド環境で実行する際に発生する例外を処理する機能を持つ．もとのスタンドアロンのアプリケーションロジックは，通常，分散処理に関連する例外を処理する機能は持っていない．*Exception handler* は，これらの例外を実行時に捕捉し，アプリケーションロジックの手を煩わすことなく処理する．あるいは，アプリケーションロジックが処理可能な例外へと変換してアプリケーションロジックに渡すなどの処理を行う．

Scheduler オブジェクトは，サーバのメソッドを同時に実行する複数のスレッドの同期制御を行う役割を持っている．*TRMI* を用いると，１つのサーバを複数のクライアントから同時アクセスすることが容易に可能となる．それは，シングルスレッド環境用に設計されたアプリケーションロジックが，複数の計算機から構成されるマルチスレッド環境で実行されることに他ならない．複数のクライアントによる同時アクセスは，何らかの競合状態を発生させる場合がある．*Scheduler* オブジェクトは，複数のクライアントからの同時アクセスにより発生する複数のスレッドの同期制御を行うことにより，サーバのメソッドが安全に実行できるようにする役割を持つ．

Factory オブジェクトは，分散環境におけるサーバオブジェクトの生成や配置の機能を持つ．もとのスタンドアロンのアプリケーションロジックは，サーバとクライアントが同一のホスト上で実行されることを前提としている場合がほとんどである．また，サーバとクライアントは共に１つずつ存在することを仮定している場合も多い．しかし，*TRMI* を使用することにより，サーバやクライアントは，以上のような位置的あるいは数量的な制約から解放される．サーバとクライアントは，異るホスト上に存在したとしても，それらのホストがネットワークで接続されていれば問題ない．複数のサーバやクライアントが複数のホスト上に存在し，お互いに連携しあって共通の処理を実現する場合もある．サーバは，クライアントが開始される前に生成される場合もあるし，クライアントからの要求に従って生成されることもある．サーバは，クライアントからの要求に従って，あるいは，自分の判断で，他のホストへ移動するかもしれない．いずれの場合にしても，サーバの位置はクライアントの位置により制約を受けることはない．*Factory* オブジェクトは，サーバを生成するホストを決定し，そのホスト上にサーバを生成する．

3.1　分散化の形式

スタンドアロンシステムを分散システム化すると言っても，様々な方式が考えられる．*TRMI* の分散システムへの変換の方式は，基本的に 2 つに分類できる．第 1 の方式は，図 2(1) に示すように，スタンドアロンプログラムとして正しく動作するシステムのオブジェクトを分散処理部品と組み合わせてネットワーク経由で別の計算機からアクセスできるようにすることにより，元のシステムと等価な分散システムを開発する方式である．言わば，もともと一体のシステムをネットワークを用いて複数の計算機上に分割して協調動作させる方式である．この方式を分離型と呼ぶ．図 2(1) は，分散処理部品と組み合わされたスタンドアロンのサーバ Y が，クライアント X からネットワーク経由でアクセスできるように分散化される様子を表

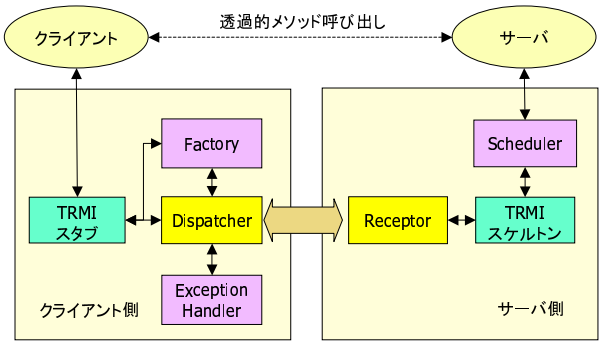


図 3: *TRMI* を用いた分散システムのアーキテクチャ

している．

第 2 の方式は，逆に統合型と呼ぶ．この方式では，図 2(2) に示すように，まず開発対象の分散システムをいくつかのスタンドアロンのサブシステムとして開発する．そして，それらのサブシステム中のオブジェクトを分散化して複数のサブシステムに共有させ，サブシステムが協調動作できるように統合する方式である．図 2(2) は，オブジェクト Y をアクセスする 2 つのサブシステムが，Y が分散化されることにより統合される様子を表している．

上記の 2 つは，ひとつのオブジェクトを分散化する場合を例として示しているが，複雑なシステムにおいては，ひとつのオブジェクトをネットワーク経由でアクセスできるようにするだけでは不十分で，システム中の複数のオブジェクトを同時に分散化させる必要が出てくる．その際には，各オブジェクトのシステム中での役割や位置づけに応じて，分散化の方式を決定する必要がある．従って，システム全体としては，分離型あるいは統合型を組み合わせた方式となる．

なお，現在の *TRMI* では，分離型あるいは統合型のどちらを用いて分散システムを構築するかは，開発者に任せている．これは，*factory* オブジェクトを用いた配備機能により制御できる．

3.2 *TRMI* の分散化手法

Java RMI などのような既存の分散オブジェクトフレームワークでは，スタブとスケルトンと呼ばれるオブジェクトが，クライアントがサーバのメソッドをネットワーク経由で呼び出す際に，メソッド呼び出しを中継する機能を有していた．RMI などの既存の方式の問題点は，これらのスタブやスケルトンといったオブジェクトを，クライアントやサーバが直接アクセスしなければならない点である．そのため，スタンドアロンのアプリケーションロジックを分散化しようとすると，どうしても，スタンドアロンの場合は必要がなかったスタブやスケルトンへのアクセスを行うコードの追加や，それに伴う他の部分の変更が発生してしまう．

TRMI でも RMI などの既存の分散オブジェクト技術と同様，*TRMI* スタブと *TRMI* スケルトンが存在する．*TRMI* では，スタンドアロンのアプリケーションロジックを *dispatcher*，*receptor*，*exception handler*，*scheduler* および *factory* といった部品と組み合わせて使用することにより，分散オブジェクト化する．クライアントおよびサーバと，これらの部品を透過的に結合させる役割を果たしているのが，*TRMI* スタブと *TRMI* スケルトンである．

図 3 に *TRMI* を用いて開発した分散システムのアーキテクチャを示す．*TRMI* はプロキシパターン [2] に基づいている．一般に，プロキシとは，サーバと同一の外部インタフェースを持つオブジェクトである．即ち，サーバのメソッドと同じ名前のメソッドを持ち，それらのメソッドの引数の型も，サーバの同名のメソッドと同一である．これらのメソッドを代理メソッドと呼ぶ．クライアントは，プロキシの代理メソッド

をサーバのメソッドと思ってアクセスする．代理メソッドの基本機能は，クライアントからのサーバのメソッド呼び出し要求をサーバに中継（委譲）し，その実行結果をクライアントへ返すことである．しかし，代理メソッドでは，サーバのメソッドを呼び出す直前と，実行終了直後に独自の処理を追加して行なうことが可能である．これにより，サーバとクライアント間にプロキシを挿入することにより，サーバに無かった様々な機能を追加することができる．

TRMI スタブはクライアント側のプロキシで *TRMI* スケルトンはサーバ側のプロキシである．*TRMI* スタブと *TRMI* スケルトンは *trmic* と呼ばれる *TRMI* が提供するツールによりサーバのソースコードあるいはバイトコードから自動生成される．

TRMI スタブはサーバと同一の外部インタフェースを持つのみでなく，同一のクラス名を持つ．従って，クライアントは，*TRMI* スタブをサーバと思ってアクセスする．これが透過的リモートメソッド呼び出しの基本である．しかし，*TRMI* スタブとサーバは混同の恐れはない．それは，サーバは *trmic* により別のクラス名に変更されるからである．

3.2.1 透過的リモートメソッド呼び出し

クライアントがサーバのメソッドを呼び出すと，代わりに *TRMI* スタブの代理メソッドが呼び出される．*TRMI* スタブの代理メソッドは，*dispatcher* に対して *TRMI* スケルトンの代理メソッドを呼び出すように依頼する．*Dispatcher* は，*receptor* を経由して *TRMI* スケルトンのメソッドを呼び出す．

さらに，サーバ側では，*TRMI* スケルトンが *scheduler* のメソッドを呼び出す．*Scheduler* もサーバのプロキシであり，サーバの代理メソッドを持つ．*Scheduler* の代理メソッドを実行するスレッドと，それが呼び出すサーバのメソッドを実行するスレッドは同一である．従って，*scheduler* の代理メソッドを実行するスレッドを制御することにより，サーバのメソッドを実行するスレッドを制御することができる．

最も単純なスレッド制御は，代理メソッドをすべて *synchronized* メソッドとして，逐次実行することである．それ以外にも，FIFO の順に実行したり，完全に並列に実行したりといった制御が，それに応じた *scheduler* を記述することで可能となる．

サーバのメソッドをネットワーク経由で実行する際に，ネットワークで問題が発生する可能性がある．発生した例外は，*dispatcher* により捕捉され，*exception handler* へ送られる．*Exception handler* は，これらの例外を，アプリケーションロジックの手を煩わすことなく処理したり，アプリケーションロジックが処理可能な例外へと変換してアプリケーションロジックに渡すなどの処理を行う．

3.2.2 配置制御

クライアントがサーバ，正確にはサーバクラスのインスタンス，を生成しようとした場合でも，実際に生成されるのは，*TRMI* スタブのインスタンスとなる．*TRMI* スタブのコンストラクタは，*factory* に対して，サーバを生成するように要求を出す．*Factory* は，*receptor* と *dispatcher* を経由して，*TRMI* スケルトンに対して，サーバを生成するように要求する．

サーバを生成する要求を受けとるホストを決定するのも，*factory* の役割である．*Factory* が全く独立して自分で判断しても良いし，*TRMI* スケルトンに相談して決定することもできる．相談を受けた *TRMI* スケルトンは，*scheduler* に判断を委譲する．具体的には，*scheduler* の *consult* メソッドを呼び出して，その戻り値としてホストを決定する．従って，*consult* メソッドを定義しておけば，自由にサーバを生成するホストの位置を制御することができる．

例えば，統合の形でシステムを開発する際には，統合対象のサブシステムの内部では個別に生成していたサーバを，システム全体で１つだけ生成する形に変更する必要がある．その場合には，サーバ生成要求は，最初の要求で実際にサーバを生成し，それ以後の要求では，サーバを新たには生成せずに，既存のサーバの参照を返すといった処理を *factory* に行わせることにより，実現できる．なお，分離型でシステムを開発する際に，１つのサーバに対して複数のクライアントのアクセスを許すような場合でも，同様な処理が *factory* で必要となる．

また，行列計算のような科学技術計算を行うようなプログラムには，同一の計算処理を異るデータに対して複数回行い，その結果を統合して最終的な計算結果とするものが存在する．このようなプログラムでは，計算処理を一つのオブジェクトに行わせるように記述しておき，そのオブジェクトを複数生成して同時に実行させるような形で記述すれば，メモリ共有型のマルチプロセッサシステムでは効率良く実行することができる．そのようなプログラムを，*TRMI* を用いて分散化し，各オブジェクトを別のホスト上に配置するような *factory* オブジェクトと結合すると，グリッド化することができる．すると，システムの拡張性を格段に改善することができる．

3.3 部品の開発

TRMI では，スタンドアロンのアプリケーションロジックと組み合わせて使用することのできる部品をユーザが自由に開発できる．しかし，完全に自由に開発していたのでは，部品間の整合性が取ることも難しい．特に，*TRMI* スタブやスケルトンとの整合性の確保が重要である．また，多くの部品に共通な処理も存在するため，それらをいつも個別に開発していたのでは効率が悪い．そこで，*TRMI* では，標準の部品を自動生成し，そのサブクラスとして，ユーザが自分自身の部品を定義できるような方式を採用した．

trmic は，*TRMI* スタブとスケルトンに加えて，標準の *exception handler*，*scheduler* および *factory* を自動生成する．これらを，それぞれ，*super exception handler*，*super scheduler* および *super factory* と呼ぶ．例えば，標準の *scheduler* では，すべての代理メソッドは，サーバのメソッドを逐次呼び出しをするような形で実装されている．ユーザがこれを取りやめ，自分自身でスレッドの同期制御を行いたい場合には，ユーザの *scheduler* の代理メソッド中で，サーバのメソッドを呼ぶ前後に必要なスレッドの同期制御の処理を行うことにより，これが実現できる．

また，標準の *exception handler* では，リモートメソッド呼び出しに起因する例外を，クライアントがもともと処理可能な例外へと変換する．例外の対応付けは，標準の *exception handler* を *trmic* を用いて生成する際に，*trmic* の GUI を用いて行う．ユーザの *exception handler* は，例外を対応づけるだけではなく，*exception handler* 内部で処理してしまって，クライアントへは知らせないなどの方式で，実装することもできる．

さらに，これらの標準の部品は，標準的な機能を持つばかりでなく，ユーザの部品で使用できるような機能を用意している．例えば，標準の *factory* では，サーバを作成するメソッドや既存のインスタンスを削除するメソッド，各サーバホストの負荷を計算するメソッドなどが用意されており，これらを用いることにより，ユーザが容易に自分の *factory* を定義できる．

また，*trmic* はユーザが自分で記述する部品の骨格となるコードを自動生成する処理も行う．骨格となるコードには，各部品が空の代理メソッドを持つ形で記述されている．これにより，ユーザが自分の *exception handler*，*scheduler* および *factory* を記述する際には，白紙の状態から記述を始めるのではなく，*trmic* が生成するユーザ部品の骨格コードの代理メソッドに本体処理を追加記述するのみで，自分の部品が作成できる．

3.4 *trmic* のカスタマイズ

trmic 自身をカスタマイズして，自動生成する部品を変更することも可能である．*trmic* は，部品を生成する時に，テンプレートとなるファイルを読み込み，それを分散化処理の対象となっているクラスに応じて書き換える形で，部品を自動生成する．従って，このテンプレートファイルを交換することにより，生成する部品を全く独自のものに変更することが可能となる．

サブクラスによるカスタマイズでは，サーバクラスごとの部品のカスタマイズが可能である．一方，通信プロトコルの変更などのように，すべてのクラスに共通な変更が発生する場合もあり，クラスごとに個別にサブクラスを定義していたのでは，手間がかかる．そのような場合，*trmic* を変更して，一括して生成する部品を変更することができる．

3.5 開発支援ツール

現在の *TRMI* は3つの開発支援ツールを提供している．第一のツールは *trmic* である．*trmic* は分散化したいサーバのクラスを入力として，標準の部品群を自動生成すると同時に，ユーザ部品の骨格となるコードを自動生成する．

第二は，*TrmiEditor* というツールである．このツールは，*trmic* が自動生成するユーザ部品の骨格となるコードを編集するためのテキストエディタである．各サーバクラスの代理メソッドごとに編集を行うことが可能である．

第三の *TrmiMonitor* というツールは，サーバが稼働しているホストの状況を一括して管理するためのコンソールである．これを用いることにより，サーバの配置位置を判断することもできるし，また，正しく配置機能が動作しているかを判断することもできる．

3.6 今後の展望

これまでの研究で，かなり多くのシステムが我々の手法で分散化できるようになったと考えられる．しかし，これまでの手法は，必要最低限の分散処理機能を *TRMI* が提供し，アプリケーション固有の部分については，その開発をユーザ，即ち分散アプリケーションシステムの開発者，に任せてきた．今後の研究の方向としては，これまでユーザに任せきりであった部分の開発を，*TRMI* としてできる限りサポートあるいは自動化していくことが挙げられる．

そのひとつの例として，ユーザの *scheduler* やユーザの *factory* の開発支援ツールの充実を検討している．スレッド制御とは，実行中のスレッドの状態遷移の制御である．従って，プログラムで直接ユーザの *scheduler* を記述するのではなく，状態遷移図などの形式でスレッドの制御方式を記述し，それをもとにコードの自動生成を行う方式などを検討中である．

オブジェクトの配置制御に関しても，オブジェクトの存在可能なホストをノードとする状態遷移図によって表現可能であると考えている．オブジェクトが各マシンに生成され，あるいは実行を開始するたびに，各ホストの状態が変化する．このような状態の遷移を図で表現することにより，ユーザの *factory* のコードを自動生成する方式を検討したいと考えている．

また，ひとつのスタンドアロンのアプリケーションロジックを分散システムへの発展させる場合には多くの方式があると考えられる．どのアプリケーションロジック中のどのオブジェクトを分散オブジェクトとするかによって，システム全体の機能や性能が大きく変化する．現状では，システムの静的あるいは動的な構造表示し，それをもとにユーザが分散化する対象となるオブジェクトを指定する形式をとっている．しかし，ユーザの判断が適切であるかどうかは判定されない．今後は，ユーザが選択した分散システムとしての構造が適切であるかどうかを判断したり，ユーザに適切な構造を推奨できるような機能の実現に取り組みたいと考えている．

参考文献

[1] 杉山安洋, TRMI によるオブジェクトの分散化, コンピュータソフトウェア, Vol.19, No.5, pp.40-59, 2002.

[2] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns : Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

4 研究発表

学会誌等

1. 杉山安洋, 加藤剛彦: “TRMI におけるマルチスレッド制御機能とその有効性の検証”, コンピュータソフトウェア, Vol. 23, No. 1, 2006(採録決定済み）.

2. 伊藤祥，杉山安洋: “ MIMO：モバイルオブジェクトのためのリモートメソッド呼び出しフレームワーク”，ソフトウェア工学の基礎 XI, 2004.

口頭発表等

1. 杉山安洋, 大河原邦彦: “TRMI におけるリモートサーバのモビリティ機能の実現”, 日本大学工学部学術研究報告会要旨集, pp.47-50, 2005 年 12 月.
2. 加藤剛彦, 杉山安洋: “TRMI におけるローカルメソッドのスケジューリング機能の実現”, 情報処理学会東北支部研究会, 2005 年 3 月.
3. 伊藤祥, 杉山安洋: “モバイルオブジェクトのためのグルーピング機能を持ったリモートメソッド呼び出しフレームワーク”, 情報処理学会東北支部研究会, 2005 年 3 月.
4. 伊藤祥, 杉山安洋: “MIMO におけるリモートオブジェクトのグルーピング機能の実現”, 日本大学工学部学術研究報告会要旨集, 2004 年 12 月.
5. 加藤剛彦, 杉山安洋: “TRMI における複数粒度でのスレッド制御方式の検討”, 日本大学工学部学術研究報告会要旨集, 2004 年 12 月.
6. 坂本倫志, 杉山安洋: “クラスローダを用いたリモートオブジェクトをロードする方式の改善”, 日本大学工学部学術研究報告会要旨集, 2004 年 12 月.
7. 北川俊広，杉山安洋: “自己組織化マップを用いた Java ソースコード間類似度測定ライブラリの試作”, 電子情報通信学会　技術研究報告, 2005 年 2 月.
8. 北川俊広，杉山安洋: “ニューラルネットワークを用いたコマンド予測シェルの試作”, 電子情報通信学会　技術研究報告, 2004 年 8 月.
9. 梶原直人，杉山安洋: “複数名技術者による同時編集作業における競合の抑制と並列化の両立”, 電子情報通信学会　技術研究報告, 2004 年 8 月.

公開ソフトウェア

1. 杉山安洋: TRMI

   URL: http://www.trmi.net

   .