論　文

マルチエージェントシステムにおける並行プロセスデバッグのためのトレーサの実現

大囿　忠親†　　新谷　虎松†

Implementing a Tracer for Debugging Concurrent Processes of a Multi-Agent System

Tadachika OZONO† and Toramatsu SHINTANI†

あらまし　本研究は，マルチエージェントシステムの開発支援環境におけるトレーサの実装研究である．マルチエージェントシステムの開発においてデバッグが問題となる．マルチエージェントシステムは並行プロセスにより構成され，マルチエージェントシステムにおけるデバッグは，並行プロセスのデバッグに関連した困難な問題をもつ．本論文ではマルチエージェントシステムをデバッグするためのトレーサについて述べる．トレーサは，プログラムの実行を追跡しプログラムの実行状態に関する情報を得るためのプログラムである．トレーサによるデバッグは，プローブ効果（probe effect）と呼ばれる問題を発生させる．プローブ効果とは，デバッガによるプログラムへの干渉が，プログラムの挙動を変えてしまうことである．プローブ効果は，並行プロセスのデバッグにおいて大きな問題となる．本論文で提案するトレーサは，エージェントへのトレーサの適用によって発生するエージェントの実行遅延を原因とするプローブ効果を抑制することが可能である．本トレーサは，プローブ効果の抑制のために，デバッグ対象のエージェントの実行遅延に合わせて，デバッグ対象ではないエージェントの実行速度を調整する．

キーワード　マルチエージェントシステム，並行プロセスデバッグ，トレーサ，プローブ効果，論理型言語

1. まえがき

マルチエージェントシステムは，エージェントと呼ばれる知的で自律したソフトウェアから構成されるシステムである．それぞれのエージェントは，並行動作し互いに協調し合いながらタスクを遂行する．マルチエージェントシステムの開発において，デバッグが問題となる．マルチエージェントシステムは並行プロセスにより構成されるので，マルチエージェントシステムのデバッグは，並行プロセスのデバッグと同様な課題をもつ．一般的に並行プロセスのデバッグは困難であり，様々な研究が行われている [1].

本論文では，RXF [2]〜[4] を用いて実装されたマルチエージェントシステムにおけるエージェントをインタラクティブにデバッグするためのトレーサに関して述べる．トレーサは，プログラムの実行を追跡（トレース）し，プログラムの実行状態に関する情報を得るために用いるプログラムである．トレーサによるデバッグは，プローブ効果（probe effect）[5] と呼ばれる問題を発生させる．プローブ効果とは，デバッガによるプログラムへの干渉が，プログラムの挙動を変えてしまうことを意味する．トレーサが行う，プログラムの実行状態に関する情報の取得も，プログラムへの干渉である．プローブ効果は，逐次プロセスのデバッグにおいても問題となるが，並行プロセスのデバッグにおいても，大きな問題となる．例えば，トレーサによるプログラムの干渉が，並行プロセス間の同期を狂わせる可能性がある．すなわち，トレーサが，本来はなかったはずのバグを作ってしまう．

本論文で提案するトレーサは，エージェントへのトレーサの適用に起因する実行遅延を原因とするプローブ効果の回避を目指して実装された．本トレーサは，プローブ効果の回避のために，デバッグ対象のエージェントの実行遅延に合わせて，デバッグ対象ではないエージェントの実行速度を調整する．

† 名古屋工業大学工学部知能情報システム学科，名古屋市
Department of Intelligence and Computer Science, Nagoya Institute of Technology, Nagoya-shi, 466–8555 Japan

並行プロセスのデバッグにおける問題として，並行プロセスの複雑さ，プローブ効果，並行プロセスの非決定性による再現性のなさ，そして大域時刻の欠如が挙げられる [1]．大域時刻の欠如は，たとえプローブ効果が回避できたとしても問題になる．

並行プロセスの記述方法及びデバッグ方法については，これまで多くの議論がなされている [1]．文献 [1] で述べられているデバッグ技術の研究では，並行プログラムの非決定性やプローブ効果の回避のための研究が行われてきた．並行プログラムのためのデバッグ技術は，逐次プロセスのデバッグ技術を応用したデバッグ技術，イベントに基づくデバッグ技術，並行プロセスの制御やデータの流れを表示する技術，そして並行プロセスの静的解析に分けられる．

本研究で扱っているトレーサは，逐次プロセスのデバッグ技術を応用したデバッグ技術を用いたデバッガに分類される．本トレーサは，プログラムのステップ実行機能をもつ．ステップ実行機能とは，プログラムを 1 命令ずつ実行させるための機能である．トレーサは，プログラムを 1 命令実行したあと，そのプログラムの実行を停止させ，プログラムの実行過程を調査し，そしてプログラムの実行を再開させるという動作を繰り返す．トレーサを用いることによって，プログラマは，プログラムの実行過程を詳細に調べることができる．

本論文は，**2.**でマルチエージェントシステムのデバッグについて概観する．**3.**で，プローブ効果を回避可能なマルチエージェントシステム用トレーサの実装方式，実行例，そして評価を示す．**4.**では，本研究の関連研究と考察を述べる．本論文は，**5.**で終わりとして，まとめと今後の課題を述べる．

2. マルチエージェントシステムのデバッグ

2.1 RXF 概要

本研究では，自律した合理的なエージェントの実装のための効率的な開発環境の実現のために，RXF (Reflective Familiar) [2]〜[4] を試作した．本開発環境は，制約論理型言語 [6] による，エージェントの開発を支援する．制約論理型言語による，エージェントの開発を支援するために，制約論理型言語に，並行処理の実現のためのマルチスレッド機能そしてメッセージ通信機能や，自律エージェントの実現に利用されるメタレベルプログラミング [7] のためのリフレクション [8] 機構を追加した [3]．更に，RXF 上では，ルールプログラミング機能や，日本語の形態素解析機能などが提供されている．RXF は，実装言語として C++を用いてコンパクトに実現されている [2]．

2.2 box モデルに基づくトレーサ

本トレーサは，box モデル [9] に基づく制約論理型言語のためのインタラクティブなトレーサである．box モデルは，Prolog プログラムの手続き的な実行過程を表現したモデルであり，Prolog を代表とする論理型言語のトレーサの実現に利用できる．

図 1 は，box モデルに基づく Prolog プログラムの実行過程を表している．節データベースは，図 1 中の点線枠で示される．質問は，“?- a,b.” であるとする．この質問は，サブゴール “a” 及び “b” が同時に成立することのチェック（若しくは，証明）を意味する．box モデルではプログラムの実行を 2 入力・2 出力の box（図 1 中の box1 及び box2）の生成・消滅で表現する．box は，Call 端子から渡されたリスト L の先頭要素を評価し，成功した場合は L の残りの部分を Exit 端子から次の box の Call 端子へ渡す（図 1 の box1）．ここで，リスト L は，質問のサブゴールを要素とする．このとき新しい box が生成される（図 1 の box2）．失敗した場合は，Fail 端子から直前の box の Redo 端子に L を渡す（図 1 の box2）．このとき box が消滅する（図 1 の box2）．

本研究で実装したトレーサは，box の各入力端子（図 1 中の ①，②，そして ③）で，プログラムの実行を停止させ，プログラムの実行状況を表示する．

2.3 マルチエージェントシステムへのトレーサの適用における問題点

並行プロセス中の逐次プロセスのデバッグに，トレーサを用いることは有効であると考える．しかし，単純に逐次プロセス用のトレーサを，並行プロセスのデバッグに適用することには問題がある．問題として，プローブ効果が挙げられる．この問題の原因は，

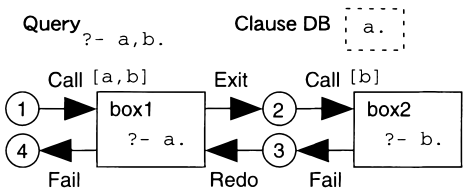

図 1 box モデルに基づく Prolog の実行過程
Fig. 1 Execution process of Prolog based on the box model.

トレーサによってデバッグ対象のプロセスが一時的に停止されてしまうことである．デバッグ対象のプロセスが一時停止している間も，非デバッグ対象のプロセスは，実行し続けている．結果として，デバッグ対象のプロセスの実行速度だけ遅くなってしまう．このようなデバッガによるマルチエージェントシステムの動作に対する影響は，好ましくない．例えば，プログラマが，帰納法によるデバッグ [10] を行うときに，バグに関する仮説を立て，その仮説を証明するという作業を行う必要があるが，このとき，システムへのデバッガの影響を考慮しつつ仮説を立てたり検証することは，プログラマにとって大きな負担となり，効率的なデバッグの妨げとなる．そのためデバッガによるマルチエージェントシステムへの影響をプログラマに意識させないようにするための工夫が必要である．

並行プロセスへのトレーサの適用には，並行プロセス中の複数の逐次プロセスを対象にする方法，及び並行プロセス中の単一逐次プロセスを対象にする方法，の 2 種類がある [1].

並行プロセスへのトレーサの適用において，並行プロセス中の複数の逐次プロセスを対象にする場合，複数のプロセスに対してそれぞれ独立のトレーサを用いる手法がある [11]．このようなトレーサは，複数のトレーサによって生成される大量のデバッグ情報の分析の困難さ，マルチウィンドウを用いてデバッグ情報を表示する際のユーザへの負担が問題となる．単一の逐次実行プロセスをトレースするだけでも，ユーザには大きな負担がかかる．なぜなら，トレーサによって生成されるデバッグ情報を理解するためには，ユーザはプログラムの実行過程を詳細に把握しておく必要があるからである．複数のプロセスのデバッグ情報をマルチウィンドウを用いて表示したとしても，それらを瞬時に理解することは困難である．結果として，そのようなトレーサは，ユーザへの負担も大きくインタラクティブなデバッグの効率改善につながりにくい．よって，本研究では，インタラクティブなデバッグという観点から，並行プロセス中の単一逐次プロセスを対象にする．

並行プロセス中の単一の逐次プロセスに対してトレーサを適用した場合，その逐次プロセスが実行を一時停止している間に，並行プロセス中の他の逐次プロセスの実行をどうするかによって，(a) トレース対象の逐次プロセスだけを一時停止する，そして (b) トレース対象の逐次プロセスだけでなくすべての逐次プロセスの実行を一時停止する，の二つのアプローチがある [1]．アプローチ (a) では，プローブ効果が問題となる．なぜならば，デバッグ対象のエージェントの実行が，トレーサによって変化してしまうからである．アプローチ (b) では，一時停止によるタイムアウトが問題となる．実際には並行動作する複数のプロセスを十分に短い時間で停止させるのは困難である [1].

本論文では，次に示す新たなアプローチ (c) を提案する．アプローチ (c) は，トレース対象のプロセスの実行は一時停止するが，そのときトレース対象ではないプロセスの実行速度をトレース対象プロセスの実行速度の低下に合わせて調整するというアプローチである．ここで調整とは，並行プロセス中の逐次プロセスの実行速度の比（以降速度比と呼ぶ）を一定に保つことを意味する．本アプローチ (c) は，アプローチ (a) の問題点であるプローブ効果の回避を試みながら，かつアプローチ (b) の問題点であるタイムアウトを避けるための新たなアプローチである．

3. プローブ効果を回避可能なマルチエージェントシステム用トレーサの実装

本章では，box モデルに基づくトレーサを並行システムに適用したときのプローブ効果の回避方法について述べる．本トレーサは，**2.3** で述べた新たなアプローチ (c) を実現したトレーサである．

3.1 基本的なアイデア

P を，n 個のプロセス $p_1, p_2, \ldots, p_n$ の集合とし，$P_{\tilde{k}}$ を，P からプロセス p_k を除いた集合とする．$p_1, p_2, \ldots, p_n$ が，時刻 t_s に実行を開始し，T 秒後の t_e に実行を終了するとする．p_i の実行速度 s_i を，1 秒間当りの p_i の命令実行数と定義する．

ある一つのプロセス p_k ($p_k \in P$) が，t_s–t_e 間のある時刻 t_0 から T_d 秒間デバッガにより実行中断されるとする．p_k は，時刻 t_0 に T_d を p_i ($p_i \in P_{\tilde{k}}$) に対して伝えたとする．通信には，T_c 秒間かかるとする．T_d を受信した p_i も，時刻 $t_0 + T_c$ から T_d 秒間実行中断したとする．$t_e - t_0 > T_c + T_d$ ならば，このときの t_s–t_e 間の p_k の実行速度 s'_k は，$s'_k = s_k \frac{T'}{T}$ ($T' = T - T_d$) であり，t_s–t_e 間の p_i の実行速度 s'_i は，$s'_i = s_i \frac{T'}{T}$ となる．このときの $p_1, p_2, \ldots, p_n$ の速度比は，$s'_1 : s'_2 : \ldots : s'_n = s_1 \frac{T'}{T} : s_2 \frac{T'}{T} : \ldots : s_n \frac{T'}{T} = s_1 : s_2 : \ldots : s_n$ となる．すなわち，p_k のデバッガによる中断時間 T_d を p_i に伝え，p_i も T_d だけ実行を中断すれば，t_s–t_e 間の速度比が保存される．

上記のような手法でプローブ効果を削減するためには，(1) p_k が時刻 t_0 に T_d を得る手法，そして (2) $T_c = 0$ にする手法，の二つの手法が必要である．(1) は，p_k が実行停止される前，すなわち遅くとも t_0 の時点で，T_d を得ることが必要であることを意味する．(2) は，t_0 から $t_0 + T_c$ の間は，速度比のずれが発生するので，速度比を保つために $T_c = 0$ が必要であることを意味する．現実的に，手法 (1), (2) の実現はほぼ不可能なので，妥協が必要である．

3.2 並行プロセスの速度比調整機構

本研究における速度比調整機構とは，マルチエージェントシステム中のエージェント間の速度比を一定に保つためのシステムである．本トレーサにおける速度比調整機構は，デバッグ対象のエージェントの実行速度の低下に関する情報を，システム中のその他のエージェントにブロードキャストして，デバッグ対象のエージェントに合わせてその他のエージェントの速度を調整することによって，速度比調整を行う．

本手法の利点は，異種エージェントから構成されるマルチエージェントシステムにも適用可能，前もってエージェントの実行速度を知る必要がない，通信における遅延時間を考慮する必要がない，大域時刻を必要としない，そして処理が簡単，の 5 点である．

更にインタラクティブなデバッグという観点から，デバッグに必要な処理は，短時間かつ少ないメモリ使用量で実行可能でなければならない．デバッグに必要な処理が，短時間かつ少ないメモリ使用量で達成できれば，デバッグ対象のプロセスに対する干渉を減少させることが可能になる．結果として，プローブ効果の回避が可能になる．

図 2 は，本トレーサのために実装した，並行プロセスの速度調整機構の概念図である．本機構は，遅延信号送信器（以降，送信器と呼ぶ）と遅延信号受信器（以降，受信器と呼ぶ）から構成される．送信器（図 2 中の円）は，トレーサに内蔵されている．受信器（図 2 中の灰色の四角）は，すべてのエージェントに内蔵されている．受信器は，エージェントを制御できるが，エージェントが，受信器を制御することはできない．

送信器は，デバッグ対象のエージェントの実行速度に合わせて，非デバッグ対象のエージェントを速度調節するのため情報（遅延信号と呼ぶ）を受信器に送信するための機構である．受信器は，遅延信号に基づいて非デバッグ対象のエージェントの実行速度を制御するための機構である．

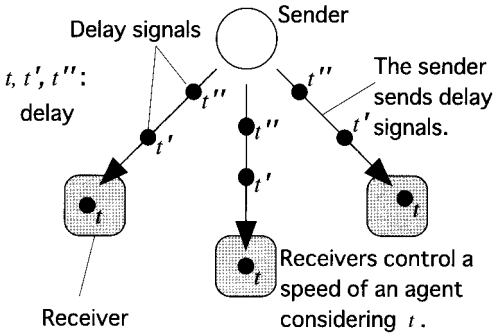

図 2 並行プロセスの速度比調整機構
Fig. 2 The speed-ratio adjustment system for concurrent processes.

デバッグ対象のエージェントは，直接トレーサにより実行制御されている．ここではトレーサは，デバッグ対象エージェントの実行の一時停止・再実行を 1 命令（本システムの場合 1 述語）ごとに繰り返す．一時停止をするタイミングは，1 命令が終了した直後である．実行再開をするタイミングは，ユーザが与えることができる．

送信器は，(1a) トレーサによるデバッグ対象エージェントの実行速度低下を検出，(1b) 受信器に遅延信号を生成，そして (1c) 遅延信号を受信器に送信，の三つの処理を繰り返す．(1b) の遅延信号は，デバッグ対象エージェントがトレーサによって一時停止されていた時間 t_1 とユーザによって与えられた時間 t_2 のうちの小さい方である．このとき，t_1 を遅延時間，そして t_2 を最大遅延時間と呼ぶ．これは，t_1 があまりにも長いと，非デバッグ対象エージェントの動作に支障が出てくる場合があるからである．例えば，エージェントが，ユーザと相互作用している場合に，エージェントが完全に停止することは不都合である．最大遅延時間による実行再開により全体の速度比にばらつきが発生し，最大遅延時間がプローブ効果の原因となる可能性があるが，上記のような理由により最大遅延時間が必要な場合もある．t_2 は，**3.4** で説明するトレースダイアログによって，いつでも変更することが可能である．

受信器は，(2a) 受信器からの遅延信号を受信，(2b) 遅延信号に基づいてエージェントの一時停止・実行再開，の二つの処理を繰り返す．(2b) の実行制御では，基本的には一時停止と実行再開までの間隔は，遅延信号をそのまま利用する．タイムアウトを引き起こす可

能性のある処理を行っている場合は，その処理の終了後に通常の処理を行う．受信器は，エージェントが，タイムアウト付きの処理待ちを行っている間，タイムアウトを引き起こす可能性があると判断する．

図 2 では，送信器は，遅延信号 t, t', そして t'' を送信した状態で，それぞれの受信器が，t を受信した状態である．このとき，t を受け取った受信器は，その受信器の制御対象のエージェントを停止させ，t 経過後に，実行再開させる．これにより，デバッグ対象エージェントで発生した実行速度の低下が，非デバッグ対象エージェントでも再現される．

ここで注意すべき点として，t はデバッグ対象プロセスのトレーサによる一時停止が終了した時点，あるいは，最大遅延時間を過ぎた後に判明する点が挙げられる．これにより遅延開始時間のずれが生じ，プローブ効果が発生する可能性がある（遅延開始時間問題と呼ぶ）．デバッガが，トレーサによる一時停止開始時と終了時に，それぞれ実行停止を表す信号（停止信号）と実行再開を表す信号（再開信号）を非デバッグ対象プロセスに送ることによって，遅延開始時間問題は軽減できると考えられる．二つの信号を用いる方法は，停止信号が届いて，かつ再開信号が届かなかった場合，非デバッグ対象エージェントが停止したままという問題をもつ．回線品質が不安定なネットワーク（例えば無線ネットワーク）上で稼動するマルチエージェントシステムのデバッグは十分考えられるが，そのような場合本方式の方が優れている．本方式では，信号がネットワーク上で失われても，停止したままという状態は発生しない．更に，本方式が遅延開始時間問題に対して脆弱であるが，実際のマルチエージェントシステムのデバッグにおいて，遅延開始時間問題はそれほど気にならないことが経験的にわかっている．以上の理由により，トレーサの実装に本方式を選んだ．

本手法では，エージェントに受信器さえ装備しておけばよいので，実行速度が不確定な異種エージェントから構成されるマルチエージェントシステムにも適用可能である．更に，通信において遅延時間があったとしても，受信する遅延信号の数は変化しないので，通信遅延を考慮する必要がない（遅延信号が失われた場合は前述のとおり）．また，実装において，大域時刻も必要としない．しかも，送信器と受信器は，少ないメモリで高速に処理できる．

本機構により，マルチエージェントシステム全体の速度比を保ちながらエージェントのデバッグを行うことが可能になる．

3.3 遅延信号受信器の実装方式

図 3 は，受信器の構成図である．図 3 は，RXF のリフレクション機能 [3] を利用した，遅延信号受信器の実装を表している．受信器は，メタレベルにおいて動作しているデバッガに機能追加することで実現される．図 3 のメタレベルと書かれた四角が，図 3 のベースレベルと書かれた四角が表すベースレベルのメタレベルである．ここでベースレベルは RXF のプログラムの実行状態を意味し，メタレベルはそのプログラムのインタプリタを意味する．メタレベルでは，エージェントポートを利用してベースレベルでのプログラムの実行を観察・制御する．エージェントポートは，RXF でリフレクションを利用するために用いられる [3]．ベースレベルの状態の取得は，ベースレベルの情報をメタレベルで扱える表現形式（メタレベル表現）に変換するエージェントポートの機能によって実現される．ベースレベルの制御は，メタレベル表現に基づいてベースレベルの状態を変更するエージェントポートの機能によって実現される [3]．図 3 では，エージェントポートを用いて，ベースレベルにおける box を，box のメタレベル表現に変換している．メタレベルでは，インタプリタの実行とデバッガの実行を交互に行っている．受信器の機能は，デバッガの実行時に実行される．この機能拡張は，デバッガを実現する部分のプログラムを，デバッガと受信器を実現するプログラムに変更することによって達成される．受信器を実現するプログラムは，エージェントポートを利用して，ベースレベルにおけるプログラムの実行を制御する．このようにリフレクション機能は，デバッガの機

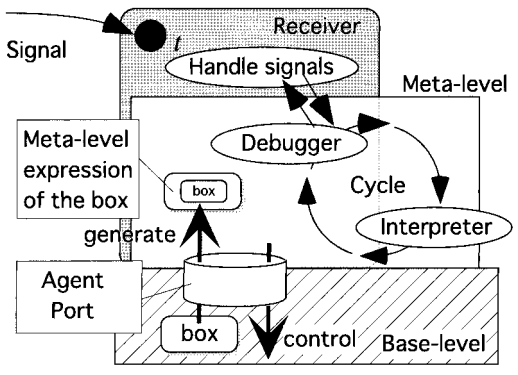

図 3　遅延信号受信器の構成
Fig. 3　The structure of the receiver.

能拡張にも利用できる．

3.4 インタフェース

トレーサの制御は，専用のダイアログウィンドウで行う（トレースダイアログと呼ぶ）．図 4 には，エージェント “a” と “b” が存在し，“a” は非デバッグ対象で，“b” がデバッグ対象である状況を表している．トレースダイアログは，図 4 中の ① とラベル付けされたウィンドウである．本トレースダイアログ上の “Creep” ボタンは，トレーサに一時停止されたデバッグ対象エージェントの実行を再開するタイミングを決定するためのボタンである．すなわち，ユーザが，“Creep” ボタンを押すたびに，デバッグ対象エージェントのステップ実行が進む．本トレースダイアログ上のチェックボックスは，本トレーサの速度比調整機能のスイッチである．本チェックボックスによって，本トレーサの速度比調整機能を有効にしたり無効にしたりすることができる．速度比調整機能が無効なときは，非デバッグ対象エージェントは実行制御されない．本トレースダイアログ上のスライドバーは，**3.2** で述べた最大遅延時間の設定に用いられる．本スライドバーを，左右にスライドすることによって，最大遅延時間を調整することができる．

② 及び ③ とラベル付けされたウィンドウは，それぞれエージェント “a” 用のインタフェース（ウィンドウ 2），及びエージェント “b” 用のインタフェース（ウィンドウ 3）である．ウィンドウ 3 には，トレーサによって出力された “b” に関するデバッグ情報が表示されている．このように，トレーサによって得られたデバッグ情報は，デバッグ対象エージェントのもつウィンドウに出力される．

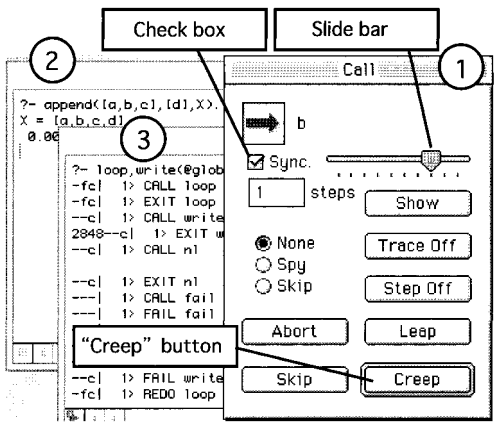

図 4　トレースダイアログ
Fig. 4　The trace dialog.

本トレースダイアログによって，本トレーサの速度比調整機能を制御することが可能になる．

3.5 評　　価

本実験の目的は，本トレーサにおける速度比調整機構が，デバッグ中のマルチエージェントシステムの速度比を一定に保つことを示すことである．本トレーサが，マルチエージェントシステムの速度比を一定に保つことが示されれば，トレーサによるデバッグ対象エージェントの速度遅延が回避されたと考えられる．すなわち，トレーサによるデバッグ対象エージェントの速度遅延に基づくプローブ効果が回避できると考える．

本実験では，エージェント A，B，そして C を，それぞれ計算機 Power Macintosh G3 350（G3），iBook（iBook），そして Power Macintosh 7300/166（7300）上で動作させる．すべての計算機は，10 Mbit/s のイーサネットでネットワーク接続されており，互いに通信可能である．エージェントが実行するプログラムは実験ごとに異なるが，一つの実験においては同じである．実験結果をわかりやすくするために，すべての実験において，エージェントは互いに実行の同期をとらないようにした．

本実験では，7 種類の実験を行った．それぞれの実験の違いは，最大遅延時間 t_2，実験に用いるプログラムの I/O 処理（I/O）の有無，そして I/O 処理等によるタイムアウト（TO）の有無等である．遅延時間 t_1 は，$t_1 = 1 \pm 0.5$（秒）で遅延信号送信ごとにランダムに決定される．すべての実験において，五つの状況における計測を行った．計測 1 では，トレーサを使用せずに実行した場合を計測した．計測 2 では，速度比調整機構を無効にしてエージェント A に対してトレーサを使用した場合を計測した．計測 3，4，5 では，速度比調整機構を有効にしてトレーサを使用した場合を計測した．計測 3，4，5 では，それぞれエージェント A，B，C がデバッグ対象エージェントである．計測 1 は，結果を評価するための基準を得るための計測である．計測 1 の結果と計測 3，4，5 の結果が類似しているのが，良い結果である．それぞれの計測では，300 秒間，各エージェントの実行速度を計測した．すべての計測において，A の実行速度を 1 として結果を集計した．

実験 1 は，$t_2 = \infty$ で，I/O と TO がともにない状況での実験である．実験 1 の目的は，本速度比調整機

構がデバッグ対象にかかわらず速度比を調整することを示すことである．実験 1 で用いたプログラムは，“ハノイの塔” を解くプログラムである．

実験 1 の結果を表 1 に示す．表 1 の 2 列目は，デバッグ対象のエージェントを表し，3 列目及び 4 列目は，それぞれトレーサが有効（on）か無効（off）か，そして速度比調整機構が有効（on）か無効（off）かを表している．A，B，C 列は，それぞれの計測における，A の実行速度に対する比を表している．計測 1 と計測 2 を比較すると，トレース対象である A の実行速度が，遅くなっていることがわかる．速度比調整機構を使用した場合の結果である計測 3，4，5 と計測 1 とを比較すると，それぞれの速度比は類似しており，計測 1 における速度比が，計測 3，4，5 でも維持されたといえる．これらの結果より，本速度比調整機構が，デバッグ対象エージェントにかかわらず，速度比を維持することが示された．

図 5 は，実験 1 における計測 2 と 3 における，B と C の速度比の変化をグラフ化したものである．図 5 の “B: off” 及び “C: off” は，それぞれ計測 2 におけるエージェント B 及び C の速度比を表し，“B: on” 及び “C: on” は，それぞれ計測 3 におけるエージェント B 及び C の速度比を表す．計測 2 では，速度比が変化するが（A が遅くなっている）が，計測 3 では，速度比が一定に保たれていることがわかる．

表 1　実験 1 の結果
Table 1　The experimental result 1.

	対象	trace	ctrl	A	B	C
計測 1		off	off	1.00	0.830	0.391
計測 2	A	on	off	1.00	1.212	0.827
計測 3	A	on	on	1.00	0.858	0.415
計測 4	B	on	on	1.00	0.851	0.416
計測 5	C	on	on	1.00	0.855	0.409

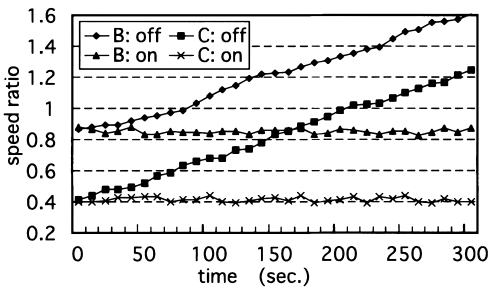

図 5　速度比の変化
Fig. 5　The variation of the speed ratios.

実験 2〜6 では，本機構を様々な条件下で評価する．実験 2，4，6 では，$t_2 = 1 \pm 0.5$ とし，遅延信号の受信ごとにランダムに変化させた．実験 3，5 では，$t_2 = \infty$ とした．このとき，実験 2，4，6 では最大遅延時間による実行再開の可能性があるが，実験 3，5 では可能性はない．

実験 2 で用いたプログラムは，実験 1 と同じ “ハノイの塔” を解くプログラムである．ただし，実験 2 では，最大遅延時間による実行再開の可能性がある．実験 3，4 で用いたプログラムは，エージェントがお互いにメッセージ通信を行い，受信したメッセージをファイルに書き出すプログラムである．実験 3，4 のプログラムでは，メッセージの内容や送り先は，ランダムに決定される．実験 5，6 で用いたプログラムは，基本的に実験 3，4 のプログラムと同じであるが，メッセージ通信に関して TO を扱う処理を含んでいる．それぞれの実験 2〜6 において，計測 1〜5 を行った．本論文では，紙面の都合上，実験 2〜6 に関しては，計測 1〜3 だけを示す．

実験 2〜6 のそれぞれの計測結果を表 2 にまとめた．表 2 の 1 列目は，I/O 処理の有無を表す．すなわち，実験 2 以外のプログラムは，I/O 処理を扱っていることを表している．表 2 の 2 列目は，I/O 処理における TO の有無を表す．すなわち，ここに $\surd$ が付けられた行の実験のプログラムは，TO を扱った．表 2 の 2 列目は，その実験における遅延時間 t_1 と最大遅延時間 t_2 を表している．A，B，そして C 列は，それぞれの計測における，A の実行速度に対する比を表している．表 2 から，本システムは，I/O や TO を扱っても，速

表 2　実験 2〜6 の結果
Table 2　The experimental results 2〜6.

	I/O	TO	t_2(秒)		A	B	C
実験 2			$t_2 = 1 \pm 0.5$	計測 1	1.00	0.828	0.390
				計測 2	1.00	1.165	0.850
				計測 3	1.00	0.954	0.511
実験 3	$\surd$		$t_2 = \infty$	計測 1	1.00	0.854	0.412
				計測 2	1.00	1.204	0.830
				計測 3	1.00	0.862	0.430
実験 4	$\surd$		$t_2 = 1 \pm 0.5$	計測 1	1.00	0.846	0.408
				計測 2	1.00	1.212	0.827
				計測 3	1.00	0.979	0.502
実験 5	$\surd$	$\surd$	$t_2 = \infty$	計測 1	1.00	0.856	0.413
				計測 2	1.00	1.224	0.769
				計測 3	1.00	0.856	0.422
実験 6	$\surd$	$\surd$	$t_2 = 1 \pm 0.5$	計測 1	1.00	0.862	0.385
				計測 2	1.00	1.232	0.847
				計測 3	1.00	1.037	0.467

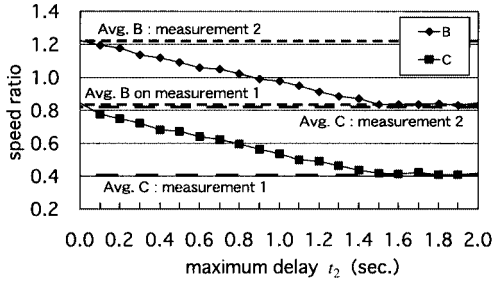

図 6　実験 7 の結果
Fig. 6　The experimental result 7.

度比を維持することが可能であるが，$t_1 > t_2$ となるとき，すなわち最大遅延時間による実行再開が生じる場合は，速度比のずれが顕著になることがわかった．なお計測 4 と 5 に関しても計測 3 と同様な結果が得られた．

実験 7 では，t_2 の変化による速度比のずれを評価する．実験 7 で使用したプログラムは，実験 1 と同じで，I/O や TO の処理はない．ここでは，t_2 の値を 0 から 2 まで 0.1 ずつ変化させながら計測 1〜5 を行った．t_2 の値が大きいほど最大遅延時間による実行再開が発生する確率が低くなる．実験 7 の結果を図 6 に示す．図 6 の縦軸は B と C の A に対する速度比を表し，横軸は t_2 を表す（単位は秒）．図 6 から，$0 \leqq t_2 \leqq 1.5$ では，t_2 が小さくなるにつれて速度比のずれが大きくなることがわかる．また，$t_2 = 0$ のときに速度比のずれが最大になり，評価 2 の値と同等になることがわかる．更に，$t_2 > 1.5$ で，ずれがなくなることがわかる．これは，t_1 の最大値が 1.5 なので，$t_2 > 1.5$ のときは $t_2 > t_1$ となり，最大遅延時間による実行再開が発生しないからである．なお計測 4 と 5 に関しても計測 3 と同様な傾向が見られた．実験 7 より，最大遅延時間が小さいと速度比のずれが大きくなることがわかった．

実験 1〜7 から，本研究で提案したトレーサを適用しても，それぞれのエージェントの速度比はそれほど変化しないことがわかった．これにより，本速度比調整機構は，デバッグ中のマルチエージェントシステムの速度比を一定に保つことができるといえる．ただし，最大遅延時間による実行再開に対する対応が不十分であると考えられる．

4. 関連研究と考察

プログラムの品質を向上させるための方法として，プログラム試験 [12] がある．プログラム試験では，プログラムの誤りを発見することによって，プログラムの品質の向上を目指す．並行プロセスのプログラム試験は，困難である [12]．同様に，並行プロセスの一種であるマルチエージェントシステムにおけるプログラム試験も，困難であると考えられる．マルチエージェントシステムは，動的にシステム構成を変更できるようなシステムの構成を目指しているので，マルチエージェントシステムにおけるプログラム試験は，非常に困難になることが予想される．なぜならば，動的にシステム構成を変更することによって，試験すべき状態数が爆発的に増加するからである．プログラム試験だけでは，マルチエージェントシステムのバグを十分に発見することは困難だと思われるので，マルチエージェントシステムの新たなデバッグ技術が必要となる．

並行プロセスにおけるデバッガとしては，アルゴリズミックデバッギング [13] のように自動的にバグを発見できるような手法の方が好ましい．なぜならば，トレーサなどを用いたデバッグをするためには，プログラマがデバッグ対象のプログラムに関する十分な知識をもっている必要があるからである．一般的な，トレーサなどの逐次プロセス用のデバッガは，並行プロセスの非決定性や大域時刻の欠如などの問題に対して本質的な解決策を示すことができない．一方，並行プロセスを静的に解析することによって，並行プロセスのデバッグにおける問題点を解決することも可能である．例えば，Krinke は，マルチスレッド化されたプログラムを静的に解析する手法を提案している [14]．しかし，一般的にこれらの手法は，高い計算量という問題点をもつ [1], [14]．すわなち，これらの手法を用いても，現実的には完全にバグを発見できるという保証はない．よって，本システムのようなデバッグ支援も有効であると考える．

並行プロセスへのインタラクティブなトレーサを考えた場合，複数の逐次プロセスを同時にトレースするよりも，並行プロセス中の単一の逐次プロセスをトレースした方が良い結果が得られる．これは，トレーサ利用時のユーザへの負担の大きさが理由として挙げられる．複数のトレーサをマルチウィンドウを用いて制御するのは，経験上，決して効率的であるとはいえない．なぜならば，複数のトレーサによって生成される大量のデバッグ情報を瞬時に理解するのが困難だからである．本アプローチでは，プログラマは，基本的に一つのエージェントのデバッグ情報だけを理解すれ

ばよい．よって，本アプローチは，本トレーサのような，インタラクティブなデバッグに適している．

5. むすび

本論文では，マルチエージェントシステムのような並行プロセスのインタラクティブなデバッグに有効なトレーサの実装方式を提案した．従来のトレーサは，並行プログラムへの適用に際し問題があった．本手法を用いたトレーサは，並行プログラムのデバッグにおける問題の一つであるプローブ効果を回避することが可能である．本トレーサは，プローブ効果の回避のために，デバッグ時に複数プロセス間の速度比を一定に保つ．本トレーサが有効なケースとして，マルチエージェントシステム中の1エージェントに対して帰納法によるデバッグを行う場合が挙げられる．

次に，速度比調整機構に基づくトレーサの実装を示した．本速度比調整機構は，遅延信号に基づく手法を用いて速度比の調整を行う．本手法は，対象とするプログラムや計算機環境を前もって知る必要なしに適用できる．更に，本手法は，大域時刻を必要としないという特長もある．実験によって，本手法がデバッグ時のマルチエージェントシステムの速度比調整を可能にすることを示した．本システムにより，プローブ効果が回避される．実験結果により，本システムが，マルチエージェントシステム中の単一エージェントのデバッグに有効であることを示した．

本アプローチは，box モデルに対して非依存であり，一般的な並行プロセス中の逐次プロセスのトレースにも適用可能である．逐次プロセス用のトレーサを，本アプローチに基づいて拡張することは，容易である．本アプローチは，並行プロセス中の単一の逐次プロセスのトレーサの実装に容易に適用でき，かつ有効な手法である．

文　献

[1] C.E. McDowell and D.P. Helmbold, “Debugging concurrent programs,” ACM Comput. Surv., vol.21, no.4, pp.593–622, 1989.

[2] 大囿忠親，新谷虎松，“マルチエージェントシステムのための制約論理型言語 RXF の実現，” 情処学論，vol.37, no.10, pp.1765–1772, 1996.

[3] 大囿忠親，新谷虎松，“自律的エージェントのための制約論理型言語 RXF におけるリフレクション機構の設計とのその実装，” 情処学論，vol.38, no.7, pp.1361–1369, 1997.

[4] T. Ozono and T. Shintani, “On a programming environment for building reflective agents,” Proc. IPSJ Int. Symposium on Information Systems and Technologies for Network Society, pp.303–309, 1997.

[5] J. Gate, “A debugger for concurrent programs,” Proc. Workshop on Parallel and Distributed Debugging, ACM, pp.130–140, 1988.

[6] J. Jaffar and J.-L. Lassez, “Constraint logic programming,” Proc. 14th ACM Symposium on Principles of Programming Languages, pp.111–119, 1987.

[7] P. Maes, “Issues in computational reflection,” in Meta-Level Architecture and Reflection, pp.21–35, Elsevier Science, North-Holland, 1988.

[8] B.C. Smith, “Reflection and semantics in lisp,” Proc. 11th ACM Symposium on Principle of Programming Languages, pp.23–35, 1984.

[9] W.F. Clocksin and C.S. Mellish, Programming in Prolog, Springer-Verlag, 1981.

[10] 長尾　真（監訳），ソフトウェア・テストの技法，第7章，pp.145–148, 近代科学社，1980.

[11] J.E. Stromme, “Integrated testing and debugging of concurrent software systems,” INDC96, 1996. http://www.kvatro.no/products/chipsy/pilot/paper-indc96/paper.html

[12] 古川善吾，伊東栄典，片山徹郎，“並行処理プログラムの試験，” 情処学論，vol.39, no.1, pp.7–12, 1998.

[13] E. Shapiro, Algorithmic Program Debugging, The MIT Press, 1983.

[14] J. Krinke, “Static slicing of threaded programs,” PASTE ’98, ACM, pp.35–42, 1998.

（平成 11 年 8 月 30 日受付，12 年 1 月 24 日再受付）

大囿　忠親　（正員）

2000 名工大工学研究科博士後期課程了．同年同大学知能情報システム学科助手．工博．エージェント，事例ベース推論の研究に従事．

新谷　虎松　（正員）

1982 東理大大学院理工学研究科修士課程了．同年富士通（株）国際情報社会科学研究所入所．知識情報処理，論理プログラミングなどの研究に従事．1993 名古屋工業大学知能情報システム学科助教授．1999 同教授．1999〜2000 米国カーネギーメロン大学ロボティクス研究所客員研究員．工博．分散人工知能，マルチエージェントシステム，意思決定支援システムの研究に従事．人工知能学会評議員．本会 AI 研究会専門委員．人工知能学会 FAI 研究会連絡委員．