(d) 情報

モバイルエージェントシステムのデバッグの問題点とその対応

Approach for Debugging of Mobile Agent System

尾崎 槙 † 東野 正幸 † 高橋 健一 † 川村 尚生 † 菅原 一孔 †

Shin Osaki † Masayuki Higashino † Kenichi Takahashi † Takao Kawamura † Kazunori Sugahara †

† 鳥取大学大学院工学研究科情報エレクトロニクス専攻

1 はじめに

モバイルエージェントシステムとは，エージェントと呼ばれる自律的なプログラムがネットワークに接続されたエージェント実行環境であるノードを移動しながら，他のエージェントと協調することにより問題を解決していくシステムである．このエージェントの移動やエージェント同士の協調といった特徴により，人間にわかりやすい擬人化されたモデルで分散システムを構築可能であり，これまでに様々な研究が行われてきた．[1, 2]． しかし一方で，これらの特徴はモバイルエージェントシステムのデバッグを難しくしており，一般的なシステムのデバッガでは対応できない．このため，モバイルエージェントシステムの開発やデバッグの支援を行う手法はいくつか提案されている．しかし，いずれもエージェントの移動に十分に対応したものではないため，モバイルエージェントシステムのデバッガとしては十分とは言えない．

そこで本研究ではモバイルエージェントの移動により発生するデバッグの問題を考察し，その問題を解決するデバッグ機能を開発する．これにより，モバイルエージェントシステムのデバッグの困難性の軽減を図る．

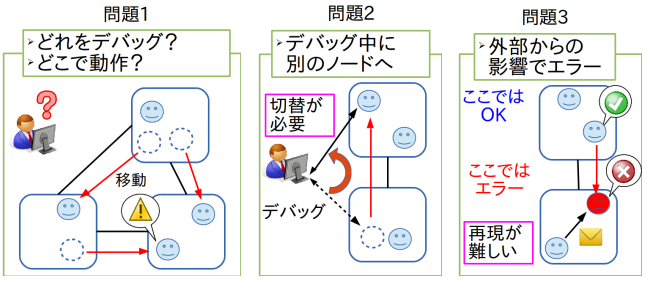
2 関連研究

システムを静的解析する手法の一つである形式手法をモバイルエージェントに応用した研究として，Mobile Object-Z (MobiOZ)[3]，LAM (Logical Agent Mobility) [4] がある．これらは，モバイルエージェントの仕様記述を SPIN (Simple Process meta language INterpreter) モデルチェッカ [5] や CADP (Con-struction and Analysis of Distributed Proceses) tool-box [6] といった検査器でモデル検査する手法を提案している．これらの手法では，モデル検査により検証されたエージェントの仕様記述から，エージェントのプログラムコードを機械的に生成することで，実装上のバグを防ぐことができる．しかし，既存システムの開発で利用するには，専用の仕様記述言語を用いて開発するか，仕様記述言語と実装言語間の変換器を新規開発する必要があり，新たなコストを要する．また，システムが大規模で複雑な場合，形式手法ではモデル検査のための計算量が膨大になり適用が難しくなる．動的解析する手法として，エージェント間の協調関係を可視化するツール [7, 8, 9, 10] が提案されている．JADE [11] や Agent Factory [12] は，エージェントの移動機能やエージェントのデバッガの一部としてマルチエージェントシステムの開発・実行環境を提供している．しかし，これらはエージェント間のメッセージ交換に焦点を当てており，移動を伴うエージェントの動作の解析には不向きである．[13] ではマルチエージェントシステムの構築を支援する統合開発環境に必要となる機能について議論しているが，エージェントの移動性については今後の展望として簡単に言及されているだけである．[14] では，各ノードに残されたエージェントの移動履歴を追跡することでエージェントの現在地を特定し，ローカルでそのエージェントの動作履歴を表示できるエージェントビューワ MiSight を提案している．しかし，監視対象のエージェントを追跡するためには，エージェントの生成時から移動ログを各ノードに記録する必要があり，生成後のエージェントの位置特定やデバッグへの利用は難しい．

3 モバイルエージェントシステムのデバッグの問題

モバイルエージェントシステムは，複数のエージェントが多数のノードに分散してノード間を移動しながら動作する特徴を持つ．この特徴により，クライアントとサーバの 2 つのプログラムを準備する必要がないこと，耐障害性が高いこと，通信処理の簡易化ができることといった様々な利点がある．

しかしこれらの特徴は，以下に示す理由によりモバイルエージェントシステムのデバッグを難しくしている．図 1 にモバイルエージェントシステムのデバッグの困難性を示す．



(a) 多数のエージェントが自律的に移動　(b) 遠隔地のノード間を移動しながら動作　(c) ノードや他のエージェントからの影響

図 1: エージェントの移動によるデバッグの問題点

問題 1. 多数のエージェントが自律的に移動
一般的には，1 つのプログラムを対象にデバッグを行えば良い．しかし，モバイルエージェントシステムでは，デバッグ対象のプログラムはエージェントとして多数存在する．このため，多数のエージェントの中からデバッグが必要なエージェントを見つけ出す必要がある．しかし，エージェントは開発者の管理下を離れて遠隔地で自律的に動作しており，各エージェントの動作状況がわからないため，どのエージェントをデバッグする必要があるか判断することが難しい．また，エージェントを指定するためには，エージェントがどこで動作しているかを把握する必要がある．しかし，ノード間を自律的に移動するため，各エージェントの位置を常に把握することは難しい．

問題 2. 遠隔地のノード間を移動しながら動作
モバイルエージェントシステムでは多数のエージェントが遠隔地のノード間を移動しながら動作する．このため，エージェントの動作確認を行うためにはエージェントが稼働するノードに対してリモート接続してエージェントの状態を確認する必要がある．また，エージェントはデバッグ中もノード間を移動するため，エージェントの移動に合わせて リモート接続先を変更する必要がある．しかし，一般的なリモートデバッグ機能ではリモート接続先が頻繁に変わることを想定していないため，頻繁にノード間を移動するエージェントに対応することができない．

問題 3. ノードや他のエージェントからの影響
モバイルエージェントはノード間を移動しながら動作するという特徴を持つ．これにより，あるノードでは正常に動作していたが，別のノードに移動した後で他のエージェントから異常なメッセージを受け取ったことで，エラーを発生させる場合がある．このように，同じ処理であっても滞在するノードやそこで動作する他のエージェントからの影響によりエラーが発生する場合としない場合が存在する．このため，エラーを再現することが難しい．

4 モバイルエージェントシステムのためのデバッグ機能

3 章で述べたエージェントの特徴に対応したデバッグを行うために問題 1 に対してはエージェントの検索機能，問題 2 に対してはエージェントに対するステップ実行機能，問題 3 に対してはエラーの再現の支援機能をそれぞれ開発することで対応する．

4.1 エージェントの検索機能

デバッグ対象は多数のエージェントであるため，デバッグするためにはその中からデバッグするエージェントを指定する必要がある．しかし，システム上には同じ種類のエージェントが多数存在するため，デバッグ対象として指定するためにはこれらのエージェントをそれぞれ識別できる必要がある．そこで，各エージェントに対して一意な識別子を付与し，これらのエージェントを識別することを可能とする．

また，遠隔地を自律的に移動するエージェントの位置を常に把握することは難しい．そこで，エージェントの検索機能によりエージェントを補足しデバッグを行う．

さらに， エージェントはノード間を移動するため，リクエストの度に再検索する必要がある．しかし，これでは無駄な通信が増える上に頻繁に移動するエージェントを捕捉できない．そこで，各ノードがデバッグ中のエージェントの移動を監視し，エージェントが移動した時に移動先のノードをデバッガに通知する．これにより，デバッグ中のエージェントはデバッガ側から位置を常に把握する．

4.2 エージェントに対するステップ実行機能

ステップ実行機能とは，プログラムの状態を確認しながら 1 命令毎に実行するデバッグ機能であり，ブレークポイント機能を拡張することで実現する．

4.2.1 ブレークポイント機能

モバイルエージェントシステムではデバッグ対象は各エージェントであるため，ブレークポイントは各エージェントのソースコードに対して設定する．エージェントをブレークする際は，デバッグ中のエージェントが動作するノードに対して設定したブレークポイントを送信する．それを受信したノードがエージェントに対してブレークポイントを設定する．そして，エージェントがブレークした時にエージェントの状態を管理元に送信することでデバッグ中のエージェントの状態を確認する．

また，デバッグ中のエージェントが別のノードへ移動した後も移動前のデバッグを継続する必要がある．そのためには，設定したブレークポイントを移動後も保持し続ける必要がある．そこで，デバッグ中のエージェントが移動する際には，エージェントのプログラムコードや実行状態に加えて設定したブレークポイントを移動する．これにより，移動雨後の継続的なデバッグを可能とする．

4.2.2 ステップ実行機能

エージェントに対してステップ実行を行う場合，ステップ実行の間にノード間を移動する場合があるため，ステップ実行の前後でリモート接続先を切り替える必要がある．4.1 節で述べたとおり，デバッグ中エージェントの位置を把握できるため，その位置を元にリモート接続先を変更可能である．

しかし，エージェントが短い間隔で連続した移動を行う場合は問題が発生する．なぜならその場合は移動先の通知が複数送信され，これらの通知はそれぞれが別の通信で行われるため，通信遅延により通知が間違った順序で届く可能性がある．これにより，エージェントの位置を誤って把握してしまう可能性がある．そこで，デバッグ中のエージェントに移動回数のカウンターを保持させる．デバッグ中のエージェントの移動時に移動先のノードと共に移動回数のカウンターをデバッガに通知することで通知の順序を判断する．これにより，移動をまたいだステップ実行を可能とし，エージェントの移動に対応したステップ実行を実現する．

4.3 エラーの再現機能

エージェントは動作しているノードやそこで動作する他のエージェントからの影響を受けてエラーを発生させるため，エラーを再現するにはエージェントの状態に加えて，エージェントが受けた影響も再現する必要がある．エージェントの状態については，エージェントはプログラムカウンタやコールスタックなどの実行状態を内部に持つため，エージェントの複製を記録することで，その時のエージェントの状態を再現することができる．

しかし，エージェントが受けた影響を再現するためには，全てのエージェントの処理を記録する必要があり，ノードやエージェントの数が増加すれば全ての処理を記録するのは現実的ではない．そこで，デバッグ中のエージェントの入出力のみを記録することでエラーの再現を支援する．エージェントは他のエージェントとのデータやメッセージのやり取りやノードのデータを読み込むことで影響を受ける．よって，デバッグ中のエージェントが読み込んだデータ，入出力を行ったノード，メッセージの送信元のエージェントなどの情報を記録することで，エージェントが受けた影響を記録できる．これにより，エージェントの状態とエージェントが受けた影響を再現し，エラーの再現を支援が可能となる．

4.4 実装

デバッグ機能を我々が開発するモバイルエージェントフレームワーク Maglog[15] に実装した．Maglog におけるエージェントは Prolog Café[16] を用いて実装されている．Prolog Café は Java で実装された Prolog から Java へのソースコード変換器及び Prolog インタプリタである．エージェントは Prolog の実行エンジンである Prolog クラスを継承した Agent クラスのインスタンスである．そして，このインスタンス毎に一意な識別子として UUID (Universally Unique Identifier) を付与することでデバッグ対象を指定する．開発したステップ実行機能で移動処理をステップ実行した結果を図 2 に示す．Maglog におけるエージェントの移動は go/1 という述語で行われる．この述語に対してステップ実行を行った結果，エージェントの移動と処理の継続が図より確認できた．よって，移動をまたがった処理でもステップ実行が継続できていることがわかる．

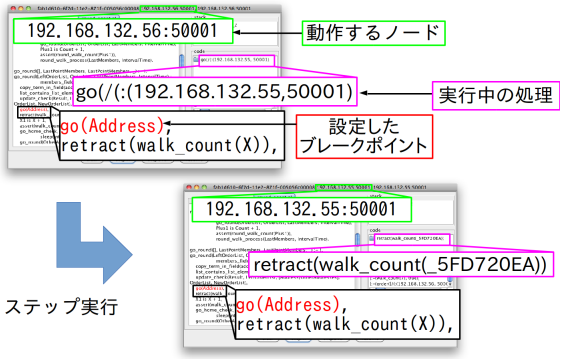


図 2: 開発したデバッガのステップ実行画面

5 評価方法の検討

一般的に行われているデバッグ手順をモバイルエージェントシステムに用いる場合，それぞれの手順で次のような問題が発生する．そして，このデバッグ手順で発生するデバッグの困難性を提案するリモートデバッガでどの程度軽減できるか評価することを考えている．

1. **エラーの存在の認識**

   (a) エージェントは遠隔地のノードで自律的に動作するため，各エージェントや各ノードの状態の把握が難しく，また，複数のノード間を移動しながら動作するため，広範囲の状態を把握する必要がある．

2. **エラー発生原因の特定**

   (a) エージェントをデバッグするには対象となるエージェントを捕捉する必要があるが，多数のエージェントが遠隔地に分散して移動しながら動作するためエージェントの位置を把握することが難しく捕捉が困難となる．

   (b) エージェントは遠隔地で動作するため，リモートデバッグ機能によりデバッグを行う．しかし，一般的なリモートデバッグ機能では頻繁な接続先の変更を考慮していないため，エージェントの移動に対応できない．

   (c) エージェントは稼働するノードや他のエージェントからの影響を受けてエラーを発生させることがあるため，エラー発生時の状態を再現することが難しくエラーの再現が難しい．

3. **修正・動作確認**

   (a) 修正を適用するにはシステムを再起動する必要がある．しかし，ノード数が増えれば再起動のコストが大きくなり動作確認が手間となる．

6 おわりに

本研究では，モバイルエージェントシステムのデバッグの困難性の軽減を目的とし，エージェントの移動にるデバッグの困難性を解決するためのデバッグ機能を開発した．今後の課題としては，更なるデバッグ機能の開発と開発したデバッグ機能が実際に有効であるか評価することが挙げられる．

参考文献

[1] Ali R. Hurson, Evens Jean, Machigar Ongtang, Xing Gao, Yu Jiao, and Thomas E. Potok. Recent Rdvances in Mobile Agent-Oriented Applications. In Albert Y. Zomaya, editor, *Mobile Intelligence: Mobile Computing and Computational Intelligence*, Wiley Series on Parallel and Distributed Computing. John Wiley & Sons, Inc., 1st edition, 2010.

[2] Abdelkader Outtagarts. Mobile Agent-based Applications : a Survey. *International Journal of Computer Science and Network Security (IJCSNS)*, Vol. 9, No. 11, pp. 331–339, 2009.

[3] Kenji Taguchi and Jin Song Dong. An Overview of Mobile Object-Z. In *Proceedings of the 4th International Conference on Formal Engineering Methods: Formal Methods and Software Engineering (ICFEM-2002)*, pp. 144–155, 2002.

[4] Dianxiang Xu, Jianwen Yin, Yi Deng, and Junhua Ding. A formal architectural model for logical agent mobility. *IEEE Transactions on Software Engineering*, Vol. 29, No. 1, pp. 31–45, 2003.

[5] Mordechai Ben-Ari. *Principles of the Spin Model Checker*. Springer London, 2008.

[6] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2010: a toolbox for the construction and analysis of distributed processes. In *Proceedings of the 17th international conference on Tools and algorithms for the construction and analysis of systems: part of the joint European conferences on theory and practice of software*, TACAS'11/ETAPS'11, pp. 372–387, Berlin, Heidelberg, 2011. Springer-Verlag.

[7] Dung N. Lam and K. Suzanne Barber. Debugging agent behavior in an implemented agent system. In *Proceedings of the Second international conference on Programming Multi-Agent Systems*, ProMAS'04, pp. 104–125, Berlin, Heidelberg, 2005. Springer-Verlag.

[8] Lin Padgham, Michael Winikoff, and David Poutakidis. Adding debugging support to the prometheus methodology. *Eng. Appl. Artif. Intell.*, Vol. 18, No. 2, pp. 173–190, March 2005.

[9] Guillermo Vigueras and Juan A. Botia. Tracking causality by visualization of multi-agent interactions using causality graphs. In *Proceedings of the 5th international conference on Programming multi-agent systems*, ProMAS'07, pp. 190–204, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] Lawrence Cabac, Till Dörges, Michael Duvigneau, and Daniel Moldt. Requirements and tools for the debugging of multi-agent systems. In *Proceedings of the 7th German conference on Multiagent system technologies*, MATES'09, pp. 238–247, Berlin, Heidelberg, 2009. Springer-Verlag.

[11] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, and Giovanni Rimassa. JADE PROGRAMMER 'S GUIDE, 2010.

[12] Rem Collier. Debugging agents in agent factory. In *Proceedings of the 4th international conference on Programming multi-agent systems*, ProMAS'06, pp. 229–248, Berlin, Heidelberg, 2007. Springer-Verlag.

[13] Simon Lynch and Keerthi Rajendran. Breaking into industry: tool support for multiagent systems. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, AAMAS'07, pp. 136:1–136:3, New York, NY, USA, 2007. ACM.

[14] 山谷孝史, 服部宏充, 伊藤孝行, 新谷虎松. モバイルエージェントのための位置透過な操作手法とエージェント分散処理への応用. 人工知能学会全国大会 (第 16 回) 論文集, 2004.

[15] Takao Kawamura, Shinichi Motomura, and Kazunori Sugahara. Implementation of a logic-based multi agent framework on java environment. In *Proceedings of IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, pp. 486–491, 2005.

[16] Mutsunori Banbara, Naoyuki Tamura, and Katsumi Inoue. Prolog cafe : A prolog to java translator system. In *Proceedings of the 16th International Conference on Applications of Declarative Programming and Knowledge Management*, pp. 1–11, 2005.