モバイルエージェントの移動に対応した動的デバッグ手法の提案

尾崎　槙[†a)]　東野　正幸[††b)]　高橋　健一[†c)]　川村　尚生[†d)]
菅原　一孔[†e)]

A Debugging Method for Tracking Mobile Agent Migration

Shin OSAKI[†a)], Masayuki HIGASHINO[††b)], Kenichi TAKAHASHI[†c)],
Takao KAWAMURA[†d)], and Kazunori SUGAHARA[†e)]

あらまし　モバイルエージェントシステムとは，モバイル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ノード間を移動しながら問題を解決していくシステムである．移動や協調と言った特徴により分散システム構築において様々な利点があるが，デバッグを難しくする要因となっている．そこで，本論文ではモバイルエージェントシステムのデバッグにおける問題点を論じ，それを解決するためのエージェント検索機能・ステップ実行機能・バグの再現機能を提案する．提案するデバッグ機能をアプリケーション開発に用いた結果，タイプ数を 41%，クリック数を 24%削減できた．

キーワード　モバイルエージェント，デバッグ，分散システム，移動，協調

1. はじめに

モバイルエージェントシステムとは，モバイル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ノード間を移動しながら問題を解決していくシステムである．エージェントの移動や協調と言った特徴は，分散システムの構築技術として期待されている [1], [2]．しかし，モバイルエージェント技術を用いたシステムが普及しているとは言い難い．この原因として，エージェントの移動によりデバッグが難しくなることが挙げられる．モバイルエージェントシステムをデバッグするためには，各エージェントの動作を把握することが重要となる．しかし，多数のエージェントが遠隔地を移動しながら動作するため，各エージェントの動作を把握することが難しい．

また，プログラムをデバッグするためには実行中のプログラムの詳細な情報を知る必要がある．このため，一般的なデバッガは遠隔ノードのプログラムに対してリモートで実行できるブレークポイント機能やステップ実行機能を提供している．しかし，エージェントはデバッグ中であってもノード間を移動する．このため，プログラムの移動を想定しないデバッガではエージェントの移動が発生するとデバッグを継続できない．

さらに，バグの原因の特定においてバグの再現性が問題となる．エージェントは他のエージェントやノードからの影響を受けるため，バグを再現するためには複数のエージェントの状態を再現する必要がある．また，エージェントの移動処理は移動時のネットワークの状態が関わる．これらの状態はプログラムを実行する度に異なる可能性があるため，同じプログラムを実行してもバグが再現しない場合があり，バグの特定が困難となる．

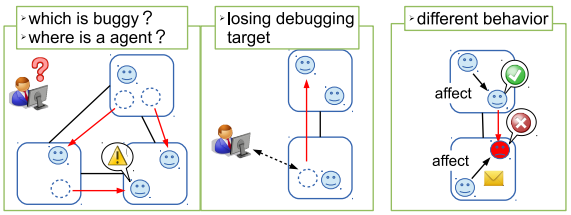
そこで本論文ではバグに関わるエージェントを発見するためのエージェント検索機能，動的に接続先を変更することで移動するエージェントに追従可能なステップ実行機能，エージェントのメッセージ交換とスナップショットの記録によるバグの再現支援機能を提案する．これにより，上記のモバイルエージェントシステムのデバッグにおける困難性の軽減を図る．

2. モバイルエージェントシステムのデバッグの問題

問題 1. 自律的な移動
モバイルエージェントシステムにおけるデバッグの対象はエージェントである．エージェントはシステム内に多数存在し，それぞれが自律的に動作すると共にノード間を移動するといった特徴を持つ（図 1a）．この特徴により，各エージェントの動作を把握することが難しくなり，バグの原因となるエージェントの発見が困難となる．このため，開発者がバグの原因となっているエージェントを見つけ出す仕組みが必要となる．

問題 2. デバッグ中の移動
遠隔ノードで動作しているプログラムの動作は，そのプログラムが動作しているノードにリモート接続して確認する必要がある．しかし，リモート接続先とは異なるノードへエージェントが移動した場合，エージェントを見失う（図 1b）．このため，エージェントの動作内容を継続

† 鳥取大学大学院工学研究科，鳥取県
Graduate School of Engineering, Tottori University, Tottori, Tottori 680-8552, Japan
†† 鳥取大学 産学・地域連携推進機構，鳥取市
Organization for Regional Industrial Academic Cooperation, Tottori University, Tottori, Tottori 680-8552, Japan
a) E-mail: s092018@ike.tottori-u.ac.jp
b) E-mail: s032047@ike.tottori-u.ac.jp
c) E-mail: takahashi@ike.tottori-u.ac.jp
d) E-mail: kawamura@ike.tottori-u.ac.jp
e) E-mail: sugahara@ike.tottori-u.ac.jp



(a) 自律的な移動 Autonomous migration. (b) デバッグ中の移動 Migration on a debugging. (c) 外部からの影響 Exterior influences.

図 1: モバイルエージェントシステムのデバッグの問題
Fig. 1 Problems on debugging a mobile agent system.

的に確認するために，エージェントの移動にあわせてリモート接続先を変更する仕組みが必要となる．

問題 3. 外部からの影響

バグの原因を確定するためには，そのバグを再現する必要がある．エージェントは他のエージェントや滞在しているノードといった周囲の状況に応じて動作が変化し，その変化もまた周囲の状況へ影響を及ぼす．さらに，状況の変化に関わったノードがネットワークから離脱する可能性もあり，モバイルエージェントシステムにおけるバグの再現は難しい（図 1c）．このため，エージェントの動作内容だけでなくエージェントが受けた影響を記録・収集し，バグの再現を支援する仕組みが必要となる．

3. モバイルエージェントシステムのためのデバッグ機能

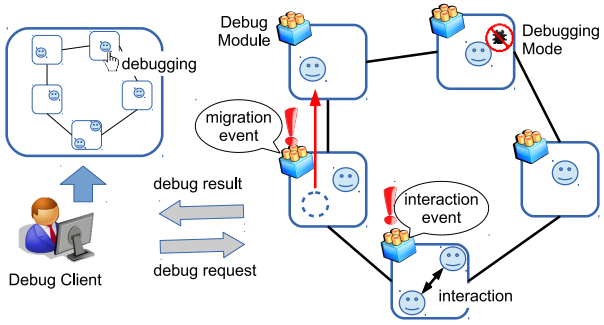


図 2: 提案デバッガの構成
Fig. 2 Structure of the debugger

提案するデバッグ環境は各ノードに設置するデバッグモジュールと開発者の元で動作するデバッグクライアントで構成される (図 2)．デバッグモジュールは設置されたノードのエージェント実行環境からエージェントの移動等のイベント通知を受けて動作する．デバッグクライアントはユーザインタフェースを持ち，開発者からのデバッグ操作を受け付ける．デバッグモジュールとデバッグクライアントは共に HTTP サーバ機能を保持しており，開発者の要求を受けたデバッグクライアントからデバッグモジュールに対して要求を送り，デバッグ処理の結果をデバッグモジュールからデバッグクライアントに送ることで各デバッグ処理を行う．

3.1 エージェント検索機能

モバイルエージェントの移動により，バグの原因となっているエージェントの発見が難しい．そこで，バグに関わるエージェントの発見を支援するためのエージェント検索機能を提案する．

モバイルエージェントシステムにおける代表的なバグにエージェントの移動によるものがある．例えば，開発者の意図しないノードへ移動する，移動するはずが移動しない，高頻度で移動を繰り返す等の異常動作が考えられる．これらの動作はエージェントの移動経路，エージェントの移動間隔等に異常な状態が現れる．そこで，これらを発見するために各ノードのデバッグモジュールに移動ログを記録する．移動ログは以下の項目で構成される．

node_src 移動元ノードの識別子
node_dest 移動先ノードの識別子
staying_time このノードでの稼働時間
agent_name エージェントの役割
agent_id エージェントの識別子

移動による異常動作は最初の 3 つの情報から判断する．*node_src* と *node_dest* はエージェントの出発・到着イベントをエージェント実行環境から受け取り記録する．このログを追跡することでエージェントの移動経路を取得する．また，*staying_time* は到着と出発のイベントの時刻の差から各ノードでの稼働時間を計算する．これらにより，移動するはずのエージェントが移動しないことや高頻度の移動を検知できる．*agent_name* や *agent_id* は開発者への検索結果の掲示や開発者による検索対象の指定に用いる．*agent_name* は開発者がつけたエージェントの名前及び役割を表す．*agent_id* はエージェントを識別するためのもので，一意な識別子を表す．

これらの情報を検索する例として NODE と NAME を用いた検索クエリの例を示す．NODE は指定したノードで動作するエージェントを検索する．NAME はエージェントの役割で検索を行う．これらの検索クエリや比較演算子等を組み合わせることでシステム内のエージェントを絞り込む．例えばノード “AAA” で動作する “get_agent” という役割のエージェントを検索する場合は以下のような検索クエリとなる．

```
NAME=get_agent & NODE=AAA
```

この検索クエリを “AAA” のデバッグモジュールに送り，検索クエリに当てはまるエージェントが見つかればデバッグクライアントに対して結果を送り返す．これにより，“AAA” で動作する *get_agent* の一覧を取得する．

また，検索クエリ ROUTE_OF により移動ログを用いたエージェントの経路検索を行う． 例えば以下のクエリは *get_agent* の移動経路を検索する．

```
ROUTE_OF(NAME=get_agent)
```

これを受けたノードは *node_src* と *node_dest* に記録されているノードに対して検索クエリの転送を繰り返し，移動したことのあるノードを収集する．これを集約してデバッグモジュールに返すことで，エージェントの移動経路を得る．これによりエージェントの移動に関わるバグの発見を支援する．

3.2 エージェントに対するステップ実行機能

一般的なプログラムのデバッグにはブレークポイント

機能やステップ実行機能を用いる．これをモバイルエージェントシステムに対して用いる場合，移動した場合でも継続的にデバッグするために，遠隔地のノード間を移動するエージェントの移動に合わせてリモート接続先を切り替える必要がある． エージェントは頻繁にノード間を移動するため，リクエストの度にエージェントを再検索することは無駄が多い．そこでまず，ステップ実行するエージェントを検索機能で見つけ出し，そのエージェントをデバッグモードとする．デバッグモジュールはデバッグモードのエージェントの移動を監視し，エージェントの移動時に *node_dest* と *agent_id* をデバッグクライアントに通知する．デバッグクライアントは通知に従ってリモート接続先を切り替える．

また，エージェントがステップ実行中であったとしても移動先ノードはそのことを知らないため，エージェントの移動が完了した直後にステップ実行が正しく行えない．このため，移動先のノードでエージェントのデバッグを継続するためには，設定したブレークポイント等のデバッグに必要な情報も移動する必要がある．そこで，デバッグに必要な情報をエージェントが移動する前に送ることで，エージェントが移動直後も移動先ノードでステップ実行を継続する．

3.3 バグの再現支援機能

バグの原因の特定においては，原因箇所を限定していくために反復的にバグを再現する必要がある．しかし，モバイルエージェントシステムでは，エージェントの移動や協調によりバグの再現が難しい．そこで，エージェント毎の動作を記録し，バグ発生時に記録したエージェントの状態を復元することで，バグの原因特定を支援する．

3.3.1 移動による影響の再現

バグを再現するためにはデバッグ中のプログラムの状態を再現する必要がある．しかし，エージェントの移動はネットワークの状態に影響を受け，再現できない可能性がある．そこで，移動時に作成するエージェントのスナップショットを記録することでエージェントの状態を記録する．エージェントの動作の再現時にエージェントの移動処理の代わりに記録した状態を用いることでエージェントの移動を再現する．

3.3.2 協調動作による影響の再現

エージェントは他のエージェントと協調動作を行い，これによって発生するバグが存在する．このため，バグを再現するためには協調動作を再現する必要がある．しかし，協調動作を再現するためには全ての協調相手のエージェントの動作を記録・再現する必要があり，現実的ではない．ここで，エージェントの協調動作はメッセージパッシングによって行われる．そこで，エージェントが送信したメッセージとその返信のメッセージをキー・バリューペアでデバッグモジュールに記録する．エージェントの動作を再現する際はメッセージ送信・受信をバグの再現機能の上で実行する．協調相手に送る代わりにバグの再現機能がエージェントの送信メッセージを奪い，それに対応する返信メッセージをエージェントに渡す．これにより，協調相手のエージェントを擬似的に再現し，協調動作を再現する．

3.3.3 利用イメージ

バグの再現支援機能は開発者がシステムに異常を感じた際に用いる．まず，デバッグ対象となるエージェントを検索機能により指定し，デバッグモードとする．デバッグモジュールはデバッグモードのエージェントを監視して前節までに述べたログを記録する．この時，これらのログをエージェントの移動回数と共に記録し，関連付けておく．開発者がエージェントの動作を確認したいと思った時に，復元したい時点の移動回数を指定してデバッグモジュールからエージェントの複製・メッセージ交換のログをダウンロードする．デバッグクライアントでは仮想的なエージェント実行環境を構築し，その上でエージェントを復元する．復元したエージェントをステップ実行等を用いて動作確認する．エージェントが移動処理を行う場合はデバッグモジュールから次の複製をダウンロードして置き換える．エージェントが他のエージェントにメッセージを送る時は実際には送信せずにデバッグクライアントが受け取り，メッセージの記録の中からその返信となるメッセージを返信する．これにより，過去のエージェントの動作を再現可能とし，バグの原因特定を支援する．

4. 評　　価

提案するデバッグ機能を我々が開発しているモバイルエージェントフレームワーク Maglog [3] に実装し，モバイルエージェントシステムによるアプリケーション開発時に利用することで提案機能の有効性を検証した．アプリケーションとしては，分散ハッシュテーブルアプリケーション (DHT) とホテル予約アプリケーション (HOTEL) を作成した．これらの開発時に発生したバグとその解決に有効に働いた機能を表 1 に示す．例えば DHT アプリケーションでは，データの取得後にデータ取得エージェントがアドレスの一覧から間違ったアドレスを取得したことでバグが発生した．

開発者はデータ取得エージェントが戻らないことからバグの存在に気づいた．そこで，エージェントがどこのノードで動作したかを確認するために，検索機能を用いてデータ取得エージェントの移動経路を確認した．これにより，想定外のノードへ移動していることが分かった．そこで，バグの再現機能を用いて想定外のノードへ移動する直前の状態に復元し，ステップ実行機能を用いて移動処理を確認した．その結果，移動先のアドレスが間違っていること，また，アドレスを取得した一覧に正しいデー

	bugs	search	stepping	reproducing
DHT	fail to store value	✓	✓	✓
	fail to get value	✓	✓	✓
	fail to join network		✓	✓
HOTEL	fail to initialize hotel agent		✓	
	loss a part of information	✓	✓	
	search wrong data	✓	✓	✓

表 1: 発生したバグ及び有効に働いたデバッグ機能
Table 1　Bugs and the useful debugging functions.

logs	types	clicks
without our debugger	869	167
with our debugger	514	128
reduction rate	41%	24%

表 2: デバッグ時のクリック数とタイプ数の削減率

Table 2 A number of types and clicks.

タが無いことがわかった．これにより，バグの原因はアドレスを管理する別のエージェントにある事を特定した．

このデバッグで確認が必要だった情報は主にエージェントの位置，異常な動作の確認，具体的な変数の値であった．提案デバッガ無しではエージェントの位置を確認するために移動処理に関するログを出力し，各ノードで出力されるログをひとつひとつ確認する必要がある．また，異常な動作と具体的な変数の値の確認ではリモートデバッグ機能を用いても，エージェントの移動によりエージェントを見失う．さらに，バグの発生後に手順を再現することが難しい．一方，提案デバッガを用いた場合は，検索機能によりエージェントの位置を，ステップ実行機能によりエージェントが移動したとしても変数の値や動作内容を容易に確認できた．さらに，再現支援機能により容易にバグを再現でき，デバッグの困難性を削減できた．

デバッグに要したタイプ数とクリック数を表 2 に示す．タイプ数は 41%，クリック数は 24%削減でき，提案するデバッガはデバッグにかかる手間を削減できた．

5. 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) [4]，LAM (Logical Agent Mobility) [5] 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 [6] や CADP (Con-struction and Analysis of Distributed Proceses) tool-box [7] 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ためには，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，モデル検査のための計算量が膨大になり適用が難しい．

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 [8] が提案されている．JADE [9] や Agent Factory [10] は，マルチエージェントシステムの開発・実行環境の一部としてデバッガを提供している．しかし，これらは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．[11] 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．

MiLog [12] 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 を提供している．しかし，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を追跡することは難しい．

6. お わ り に

本論文ではモバイルエージェントシステムにおけるデバッグの問題点を示し，それらを解決するためにモバイルエージェントの移動を考慮したデバッグ機能を提案した．アプリケーション開発時に本提案機能を用いることで，モバイルエージェントシステムのデバッグを支援できる事を示した．

謝　　辞

本研究は科学研究費補助金（26330223）の支援を受けて行なった．

文　　献

[1] A.R. Hurson, E. Jean, M. Ongtang, X. Gao, Y. Jiao, and T.E. Potok, “Recent advances in mobile agent-oriented applications,” pp.106–139, John Wiley & Sons, Inc., 2010.

[2] A. Outtagarts, “Mobile Agent-based Applications : a Survey,” International Journal of Computer Science and Network Security (IJCSNS), pp.331–339, 2009.

[3] T. Kawamura, S. Motomura, and K. Sugahara, “Implementation of a logic-based multi agent framework on java environment,” Proc. of IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems, pp.486–491, 2005.

[4] K. Taguchi and J. Song Dong, “Formally specifying and verifying mobile agents model checking mobility: the MobiOZ approach,” International Journal of Agent-Oriented Software Engineering, pp.449–474, 2008.

[5] D. Xu, J. Yin, Y. Deng, and J. Ding, “A formal architectural model for logical agent mobility,” IEEE Trans. on Softw. Eng., vol.29, no.1, pp.31–45, 2003.

[6] M. Ben-Ari, Principles of the Spin Model Checker, Springer London, 2008.

[7] H. Garavel, F. Lang, R. Mateescu, and W. Serwe, “CADP 2010: a toolbox for the construction and analysis of distributed processes,” Proc. of the 17th international conference on Tools and algorithms for the construction and analysis of systems, pp.372–387, 2011.

[8] L. Cabac, T. Dörges, M. Duvigneau, and D. Moldt, “Requirements and tools for the debugging of multi-agent systems,” Proc. of the 7th German conference on Multi-agent system technologies, pp.238–247, 2009.

[9] F. Bellifemine, G. Caire, T. Trucco, and G. Rimassa, “Jade programmer ’s guide,” 2010.

[10] R. Collier, “Debugging agents in agent factory,” Proc. of the 4th international conference on Programming multi-agent systems, pp.229–248, 2007.

[11] S. Lynch and K. Rajendran, “Breaking into industry: tool support for multiagent systems,” Proc. of the 6th international joint conference on Autonomous agents and multiagent systems, pp.136:1–136:3, 2007.

[12] Y. Takafumi, H. Mitsuhiro, I. Takayuki, and S. Toramatsu, “A location transparency method for mobile agents and application of agents ’ distributed system,” 2004.