モバイルエージェントシステムのデバッガの提案

Proposal of Mobile Agent System Debugger

太田垣 真也 † 東野 正幸 † 高橋 健一 † 川村 尚生 † 菅原 一孔 †
Shinya Otagaki † 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で接続された計算機であるノード間を動き回り，問題を解決するシステムである．これを用いることでクライアント/サーバ方式のプログラム設計に必要な通信プロトコルの決定及び通信プログラムの記述が必要ないため，分散システムのプログラミングが平易となる．また，無線による通信で通信路の切断が余儀なくされる場合，切断される前にエージェントを通信先に移動させておくことにより，通信路が切断された状態でもエージェントの実行を継続することができる．そのため，分散システムの構築技術としてモバイルエージェント技術の実用化が望まれている．

しかし，モバイルエージェントシステムにはデバッグが難しいという問題がある．その原因として，遠隔地のノード上でエージェントが動作するため，エラー等の異常な状態のエージェントの発見が遅れること，また，1 つのエージェントの修正と動作確認のためにシステムの停止と再起動が必要になり手間がかかるということがある．

そこで本稿では，異常状態にあるエージェントを発見し，システムを停止させずにエージェントの修正を行うためのモバイルエージェントシステムのデバッガを提案する．ここでいう異常状態とは，エージェントがエラーを出している状態や意図しない動きをしている状態のことを指す．提案するデバッガの概要を図 1 に示す．提案するデバッガは，定義された異常状態にあるエージェントの自動的な発見機能と指定した状態や振る舞いの条件に当てはまるエージェントの検索機能を持ち，また，システムを停止させずに修正対象となるエージェントだけを停止させてデバッグすることを可能とする．さらに，遠隔地の計算機上で起動するノードやそこに滞在するエージェントに対するデバッグを開発者が一箇所から行うことを可能とする．これにより，遠隔地にいる異常状態のエージェントを発見し，システムを停止させずに異常状態にあるエージェントを修正することができるため，モバイルエージェントシステムのデバッグの手間を削減することができる．

2 デバッガの構想

2.1 一般的なデバッグ

デバッグは，エラーの原因を突き止め，それを修正するプロセスである．一般的なデバッグの手順を以下に説明する．

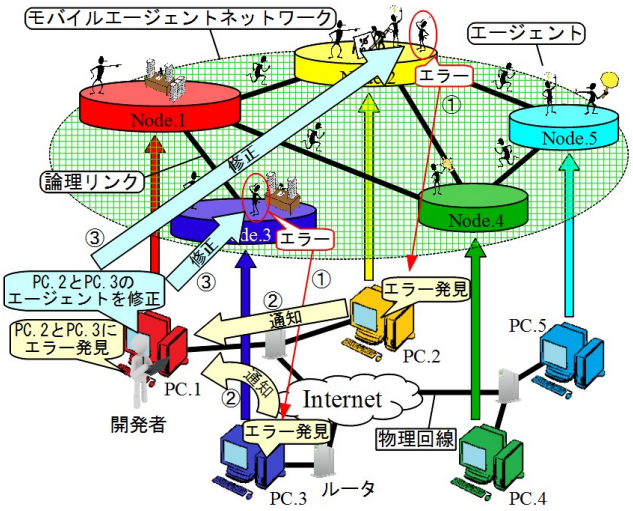

図 1: 提案するデバッガの概要

手順 1. バグの認識: 最初の手順として，エラーを見つけるという段階がある．作成したプログラムの動作を確認するためにプログラムを実行すると，エラーが出力されて実行が途中で停止されるか，想定外の結果が出力されることで，エラーの存在が発覚する．

手順 2. バグの発生源の分離: この手順では，バグが認識されたプログラムへの入力されたデータは正しいか，正しく読み込まれたか，正しく処理されているか等の検証を行う．一般的なデバッガの多くが持つブレークポイント，ステップ実行，変数確認等の機能を使いプログラムの動作や変数を確認して検証を行う場合が多い．これにより，エラーが起きている部分を絞り込むことができる．

手順 3. バグの原因の特定: 手順 2 においてエラーが起きている部分を絞込み特定した後，エラーが発生している原因を特定する．ここでのブレークポイント等の機能をもちいる場合が多い．また，バグの原因を突き止めたら，コード中の類似した箇所を調べ，同じ誤りがないかを調査する．

手順 4. バグの修正方法の決定: この手順では，特定したエラーの発生している原因をどのように修正するかを決定する．これは，既存のバグの修正によって別の新たなバグが発生してしまうことや既存のバグによって隠れていたバグが表面化する場合があるためである．

手順 5. 修正とテスト: この手順において，手順 4 で決定した方法でエラーの原因の修正を行い，その修正したプログラムを再度起動してテストを行う．このテストでエラーの修正が行えているかということと，その修正が副作用を引き起していないかを検証する．この時も手順 2 で使用したようなブレークポイント等の機能を活用し，プログラムが正常な動作を行なっているかを確認することが多い．

2.2 モバイルエージェントでのデバッグの問題

モバイルエージェントシステムは分散システム 1 つであり，複数の計算機の実行環境上でプログラムが動作する．そのプログラムはエージェントと呼ばれ自律的に動作する．また，エージェントはネットワークで接続された計算機上の実行環境であるノード間を動き回り，問題を解決するための処理を行う．モバイルエージェントシステムとはこのようなものであり，その開発に際し，デバッグの各手順において以下のような問題がある．

手順 1. バグの認識: 一般的な場合と違い，モバイルエージェントシステムのデバッグの場合は各エージェントが自律的に動作するプログラムであるため，あるエージェントがエラーを出力して停止しても他のエージェントは正常に動作し続ける．そのため，あるエージェントがエラーを出力したとしても，すぐにシステムが停止してしまうことや分かり易い異常な動作を示すことが少ない．これにより，モバイルエージェントシステムにおいては，エラーの発見が遅れるということがある．

手順 2. バグの発生源の分離: モバイルエージェントシステムの場合は，どのエージェントでエラーが発生しているかを検証し，そのエージェントのエラーが起きている部分を絞り込んでいく．しかし，モバイルエージェントシステムの場合，エージェントの処理のタイミングの問題でエラーが出る場合があり，検証のためのエラーの再現が困難である．また，一般的なプログラムのデバッグで用いられるブレークポイント，ステップ実行，変数確認等の機能をそのまま利用することができないため，エラーの原因部分の絞り込みが困難である．さらに，複数の計算機でプログラムが動作するため，一般的なシステムに比べてエラーの再現による原因の絞り込みのために停止や再起動の処理を複数の計算機に行うことは手間がかかる．

手順 3. バグの原因の特定: 一般的なデバッグと同じようにエラーが発生している原因を特定する．しかし，モバイルエージェントシステムでは，手順 2 と同様に一般的なデバッグで用いられるブレークポイント等の機能をそのまま利用することができずエラーの特定が困難である．

手順 4. バグの修正方法の決定: この手順でも一般的なデバッグと同じように特定したエラーの発生している原因をどのように修正するかを決定する．しかし，エージェントはそれぞれが自律的に動作するため，既存のバグの修正による新たなバグの発生や隠れたバグの表面化を予測することが困難である．

手順 5. 修正とテスト: モバイルエージェントシステムの場合においてもエラーの原因の修正を行い，その修正が正しいかをテストで確認する．しかし，手順 2 で述べたように，一般的なシステムに比べて停止や再起動の処理を複数の計算機に行うため，エラーの修正のためのシステムの停止や再起動は手間がかかる．また，エラーが出力された状況の再現が困難であるため，エラーの出力される状況を複数再現してテストを行うことは困難である．

上記のように，モバイルエージェントシステムのデバッグには，一般的なデバッグと比較して多くの問題がある．これらの問題の内，本稿では，エラーである異常状態のエージェントの発見の遅れとシステムの修正毎の停止と再起動に手間がかかることに注目し，解決案を示す．

2.3 デバッグの困難性の原因

モバイルエージェントシステムのデバッグが難しいという問題の原因として，遠隔地のノード上で動作している異常状態となっているエージェントの発見が遅れること，1 つのエージェントの修正のためにシステムの停止と再起動が必要になることが挙げられる．

異常状態の発見の遅れ: 発見が遅れる理由としては，エージェントが離れた場所のノード上で動作し，また，ノード間を移動するため，エージェントが異常状態になったとしてもすぐに気付くことができないということや，エージェントが異常状態になった事がすぐにシステムの停止や分かりやすい異常動作に繋がるわけではないということが挙げられる．開発者の手元でエージェントが異常状態になったことがすぐに分からないため，定期的にエージェントの異常状態の発生の有無を各ノードで確認する必要がある．

修正毎の停止と再起動: 修正毎の停止と再起動は，1 つのエージェントが異常状態になり他のエージェントは正常に動き続けている場合であってもそのエージェントの修正のためにシステム全体を停止する必要があり，また，修正の確認のためシステムを再起動しエージェントが異常状態になった状態を再現する必要があるため発生する．さらに，モバイルエージェントシステムは複数の計算機上で動作している．このため，モバイルエージェントのシステムの起動や停止の操作は複数の計算機に対して行う必要がある．

これらの理由により，モバイルエージェントシステムのデバッグに多くの手間がかかる．

2.4 デバッグの困難性への解決案

異常状態の発見の遅れ: モバイルエージェントシステムのデバッグの困難性の原因の 1 つである遠隔地の異常状態にあるエージェントの発見の遅れについては，異常状態のエージェントの開発者への通知と手元か

らの検索により解決する．エージェントの異常状態には，エージェントの内部状態が異常である状態とエージェントの振る舞いが異常な状態がある．この内，内部状態が異常な場合においては，異常な内部状態を定義しておき各ノードで滞在しているエージェントが異常な状態になっていないかを確認し，発見した場合は開発者に通知するようにすることで，異常状態にあるエージェントを自動的に発見する．一方，エージェントの異常な振る舞いとは，設計のミスによる予想外の同ルートのループ移動やあるノードでの停止等の動作のことである．ある１つのノード上で発生する異常な振る舞いであれば，エージェントの異常な状態の発見と同じように発生し次第各ノードから開発者へ通知するようにする方法が考えられる．しかし，同ルート上のループ移動等の複数のノード上で発生する異常な振る舞いについては，各ノードから通知するという事が難しい．これは，ノード間を移動するエージェントの振る舞いが異常であると判断することが難しいためである．そこで，エージェントの振る舞いが異常な場合においては，振る舞いの仕方を条件として指定しシステムに対して検索を行うことで異常な振る舞いのエージェントを発見する．

修正毎の停止と再起動: 同じくデバッグの困難性の原因の１つである１つのエージェントの修正のためにシステムの停止と再起動が必要になることについては，システムを停止させずにエージェントのコード置換を行うことで解決する．分散システムのデバッガの研究として，分散システムのエラーの原因を解析してデバッグを支援する研究 [1][2] があるが，これらの研究でも，エージェントを修正し，その修正を適応するためにはシステムの停止と再起動が必要となる．分散システムの１つであるモバイルエージェントにおけるシステムの停止と再起動の処理は大変な手間である．そこで，システムは停止させずに修正の対象となるエージェントを停止させ，修正を施したソースコードから生成されたエージェントと入れ替える．その後，入れ替えたエージェントを起動しシステムへ復帰させる．ただし，このときエージェント同士が協調しての動作，つまり依存関係に注意する必要がある．これは，依存関係にあるエージェントが突然停止してしまうと相手のエージェントも正常に動作しなくなってしまう場合が考えられるためである．そのため，修正の対象となるエージェントと依存関係にあるエージェントが存在する場合には，エージェント同士の協調動作に不具合が出ることを避ける処置が必要となる．

3 デバッガの設計

提案するモバイルエージェントシステムのデバッガは，図１のように開発者の手元から遠隔地のノードやそこに滞在するエージェントに対しての動作確認等の操作を行えるようにする．これは，開発者が手元でデバッグを行える通常のシステム開発と同じようにモバイルエージェントシステムを開発できるようにすると同時に，各ノードの元に赴いてのノードやエージェントの動作確認等の操作と比較して，デバッグの手間を削減するためである．

また，提案するデバッガは，本稿において注目するモバイルエージェントシステムのデバッグの問題の解決案としての以下の機能を持つ．

異常状態の発見の遅れ

1. 定義した異常状態のエージェントの発見
2. 指定した状態に該当するエージェントの検索

修正毎の停止と再起動

3. エージェントのコード置換

これらのデバッグの困難性を解決するための機能の設計を以下に述べる．

3.1 定義した異常状態のエージェントの発見

予め異常な状態といえるエージェントの変数の値等の内部状態を定義しておき，その情報を各ノードに配置する．各ノードで滞在しているエージェントがその定義に該当するかを確認し，異常な状態に該当する場合は開発者にその旨を通知する．これにより，異常状態にあるエージェントを自動的に発見できる．

この機能において問題となるのが，エージェントの異常状態の定義と定義に該当するかの確認のタイミングである．エージェントの異常状態の定義に漏れがあると，その異常状態に陥ったエージェントが発生してもそのエージェントの自動的な発見ができない．また，この定義は開発するシステム毎に異なる．つまり，あるシステムにおいて発生が異常といえるエージェントの状態は，別のシステムにおいては正常なエージェントの状態といえる場合があるということである．そのため，このエージェントの異常状態の定義は，開発者が定義に漏れを見つけた場合や開発するシステムに適応したものとするために容易に編集できるようにする必要がある．また，エージェントの状態が定義に該当するかの確認のタイミングについては，エージェントの内部状態が変化する毎に確認を行うことで最も早く異常状態のエージェントを発見できる．しかし，エージェントの内部状態は動作に伴い頻繁に変化するため，その度に定義に該当するかを確認していては負荷が大きくなる．この定義に該当するかの確認のタイミングについては，エージェントの内部状態の変化の都度以外としては，エージェントの移動するタイミングや時間間隔を設定した上での一定時間毎に確認する方法が考えられる．

3.2 指定した状態に該当するエージェントの検索

3.1 節の方法では，設計ミスによって同ルート上をループして移動している等のエージェントの異常な振る舞いを発見することができない．これはエージェントの振る舞いの判断ために複数の任意のノード上での

動作の情報を統合した各エージェントの動作履歴が必要になるためである．そこで，各ノードでエージェントの動作に関するログを記録しておき，エージェントの振る舞いを条件として指定しシステム内に検索をかける．検索が行われると，各エージェントの動作に関するログからエージェントの動作履歴を作成する．この動作履歴を元に検索条件に該当する振る舞いをしているエージェントがいないかを捜索する．これにより，振る舞いが異常なエージェントを発見できる．

この機能において問題となるのが，エージェントの動作に関するログと動作履歴の作成処理である．エージェントの動作に関するログについては，記録するログデータによって検索の際に条件として指定できる項目が左右されるため，できるだけ多くのデータを記録できるほうが良いが，ログの記録の処理のために大きな負荷がかかるのは望ましくない．また，エージェントの動作に関する全ての情報がエージェントの異常な振る舞いの検索に必要とは限らない．そのため，記録するログのデータの種類をエージェントの異常な振る舞いの検索の条件として必要なものに限り，ログの記録の処理の負荷を軽減することが望ましい．また，エージェントの動作履歴の作成処理は，各ノードで記録するログを集めて各エージェント毎に動作の履歴を作成するものであるが，全エージェントの動作履歴を各ノードからログを集めて作成することは大きな負荷となる．そのため，検索の際に条件として指定されたエージェントの種類や滞在したことのあるノードといった情報を元に，動作履歴の作成処理の際に扱うログの量を減らすようにする．

3.3 エージェントのコード置換

エージェントのコード置換を行うにあたり，まず，エージェントを停止させる．これは，エラーを出して停止しているエージェントを修正するような場合を除いて，エージェントは基本的に動き回っているため，コードを入れ替える際に修正対象のエージェントを見失ってしまうことを防ぐためである．次にソースコードの異常な状態となった原因部分を探し，修正する．修正したソースコードからエージェントを生成し，内部状態のデータを生成したエージェントにコピーする．そして，この生成したエージェントを元のエージェントと入れ替え，再起動させるための命令を呼び出し，エージェントをシステムに復帰させる．これにより，システムを停止させることなくエージェントの修正を行うことができる．

エージェントを停止させることにより，そのエージェントと協調して動作していたエージェントに不具合が出ることが考えられる．このエージェントの依存関係の問題についての対策の１つとして，停止させているエージェントに対する他のエージェントからのメッセージをキューに蓄えておき，エージェントがシステムに復帰し次第キューに溜まっているメッセージを処理していくという方法が考えられる．しかし，他のエージェントからのメッセージに対して即時返答しなければならない場合は，この方法ではエージェントの依存関係の問題の解決策になりえない．また，その他の対策として，停止させるエージェントと依存関係のある他の全てのエージェントを停止させるという方法も考えられるが，エージェントの依存関係を把握する術がない現状では，この方法の実現は困難である．エージェントの依存関係についての対策は必要であるが，現状として実現可能であると考えれる方法は考案できていない．

4 今後の課題

エージェントの異常な振る舞いの検索機能のために記録しておくエージェントの動作に関するログのデータの種類を限定する必要がある．そのために，モバイルエージェントシステムのデバッグの際に発生する異常な振る舞いを想定し，その異常な振る舞いを検索で発見するため記録しておくログのデータを検討する．また，エージェントのコード置換の際に修正の対象であるエージェントを停止させる必要があるが，その停止処理が修正の対象であるエージェントと依存関係があるエージェントにも影響し，エージェント同士の協調動作に不具合が出ることが考えられる．このため，エージェントの依存関係を考慮したデバッグの方法について検討する必要がある．

提案するデバッガの実装については，Java ベースのモバイルエージェントフレームワークのデバッグが行えるものを目指す．まず，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] を対象に実装を行う．また，評価の方法について検討する必要がある．

5 おわりに

我々は，モバイルエージェントシステムのデバッグの困難性を解決するために，異常な状態のエージェントを早期に発見でき，システムを停止させずにエージェントを修正できるデバッガを提案した．デバッグの際にエージェントの依存関係を考慮する必要があるが，依存関係のないエージェントのデバッグまでの考案しかできていない．今後の課題として，依存関係を考慮したデバッグの方法を考案すると共に実装し，評価することが挙げられる．

参考文献

[1] Charles Killian, James W. Anderson, Ryan Braud, Ranjit Jhala, and Amin Vahdat. Mace: Language Support for Building Distributed Systems. *PLDI*, 2007.

[2] Darren Dao, Jeannie Albrecht, Charles Killian, and Amin Vahdat. Live Debugging of Distributed System. *LNCS 5501*, pp. 94–108, 2009.

[3] Shin-ichi Motomura, Takao Kawamura, and Kazunori Sugahara. Logic-Based Mobile Agent Framework with a Concept of “Field”. *IPSJ Journal*, Vol. 47, No. 4, pp. 1230–1238, 2006.