F-008

発展型エージェントシステムのための監視アーキテクチャの設計
Design of Surveillance Architecture for Evolutional Agent System

金子 智哉† 打矢 隆弘† 内匠 逸†
Tomoya Kaneko Takahiro Uchiya Ichi Takumi

はじめに

近年，急速に進んでいる情報化社会の中で，ネットワークシステムとこれを利用し提供されるネットワークサービスの重要性が高まっている．これに伴い，新たなネットワークサービスの導入や既存のサービスの更新，改良されたサービスへの移行等が頻繁に行われるため，より柔軟なサービスを実現するエージェントシステムに注目が集まっている．その中で，ユーザやエージェント動作環境とのインタラクションを通して獲得される情報・知識・経験を活用して，システム全体の機能・性能の維持・改善を能動的に行うシステムとして，図 1 のような発展型エージェントシステムが提案されている[1][2].

発展型エージェントシステムを実現するための機能として以下の機能が必要となる．

- エージェント／エージェントシステム再利用機能
- 異種エージェント組織化機能
- エージェント組織の発展的再構成機能

本研究では，これらの機能の中からエージェント組織の発展的再構成機能に着目し，発展型エージェントシステムの実現を目指す．

エージェント組織の発展的再構成の処理は，(1)再構成の起動制御，(2)再構成の実行，(3)再構成後の評価によって行う．(2)に関しては，エージェント組織の全体・中間段階・末端の再構成プロトコル[3]が提案されているが，本研究では(1)の再構成の起動制御について検討を行う．

本研究では，構成されたエージェントシステム全体を考慮した起動制御を行うため，監視アーキテクチャの導入を提案する．監視アーキテクチャは，エージェント組織内の各エージェントに対し監視を行い，それぞれに適した起動制御を実行する．

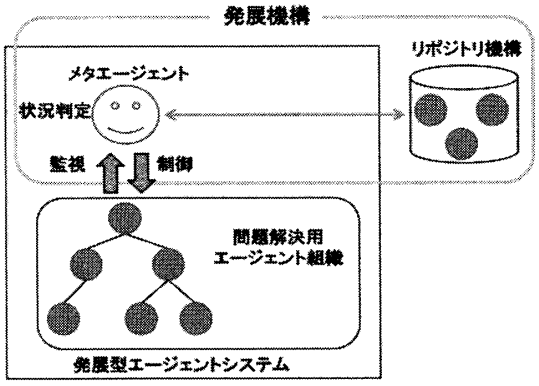

図 1：発展型エージェントシステム

†名古屋工業大学 大学院 工学研究科 情報工学専攻
Nagoya Institute of Technology, Graduate School of Engineering

本稿では，従来型のエージェントシステムにおける起動制御の問題点を説明し，その解決案として監視アーキテクチャの提案，また，監視アーキテクチャのプロトタイプ実装と実験･評価について報告する．

2．エージェントシステムの動作制御

2.1 従来型エージェントシステム

従来のエージェントシステムでは，システムの動作制御を事後対処的に行っていた．この制御手法には，以下のような問題点が存在する．

- **知識の局所性**
  各エージェントは自己の監視や制御に関するルールしか持たないため，他のエージェントやシステム全体の動作を考慮した動作制御を行うことが困難である．
- **静的な動作知識**
  エージェントは開発者の静的な記述に基づき動作，及び，その動作制御を行う．そのため，開発者の想定しない動作を行った場合や未知のエージェントを組織内に組み込む場合，適切な処理を行うことが困難である．
- **再構成の非効率性**
  組織内のエージェントに何らかの問題が発生し，組織再構成の必要性が出た場合，タスク通知・入札・落札といった三つのプロセスを経ることで，再構成を行う．各プロセス間では，それぞれ待ち時間が発生するため，全体としての再構成の完了までに時間がかかり，異常状態からの復帰が遅れる．

2.2 監視アーキテクチャ

前節における問題点を解決するため，エージェントシステム全体の動作を考慮した制御を行う監視エージェント，及び監視アーキテクチャ(図 2)を提案する．

- **監視エージェント**
  監視エージェントは，システム固有の問題解決知識を有する従来のエージェントシステムとは独立して動作し，監視対象となるエージェントの状態情報を収集し，集約された観測情報に基づき各エージェントに対する動作制御を行う．

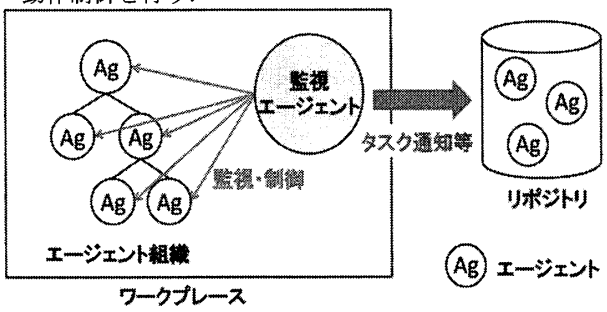

図 2：監視アーキテクチャ概略図

● **監視アーキテクチャ**

監視アーキテクチャは，従来のエージェントシステムに監視エージェントを暗黙的に付加する．また，監視対象エージェント生成時に，自動的に監視ルールの付与を行う．

3. 監視アーキテクチャの設計

監視アーキテクチャの全体構成を図 3 に示す．実装部分は大きく分けて以下の二つに分けることができる．

● **監視対象エージェントの機能拡張**

監視対象となるエージェント組織内のエージェントに対し，定期的に自己の状態情報を取得し，監視エージェントに通知する動作知識を付加する．この動作知識は，監視対象のエージェントが生成されたときに，暗黙的に付与する．これにより，既存のエージェントシステムに対して修正することなく，監視機能を付与することが可能となる．

● **監視エージェント**

監視エージェントの有する機能を以下に示す．

[1] 状態情報の収集･蓄積

各監視対象エージェントは，前述の拡張により付与された動作知識に基づき，自身の状態をメッセージとして監視エージェントに対して送信する．監視エージェントはそれらの情報を蓄積･更新し，各エージェントの状態の観測を行う．

監視エージェントは各エージェントの状態情報として，監視対象から送られるメッセージの到着間隔の揺らぎを観測する．一定間隔でメッセージを送るよう動作知識を付与しているため，監視エージェントは，本来メッセージがいつ到着するかを把握している．そのため，その差を蓄積し観測を行う．

[2] 異常検出･特定

エージェントの状態情報からシステムの異常を検出し，異常状態にあるエージェントの特定を行う．本研究では，異常検出と異常エージェントの特定を以下の状態情報に基づき行う．

- 監視対象エージェントの動作端末全体の CPU 使用率
- 監視対象エージェントの動作端末における，エージェントプラットフォーム全体の CPU 使用率
- CPU 使用率の正常状態における上限･下限
- メッセージ到着時間の遅延

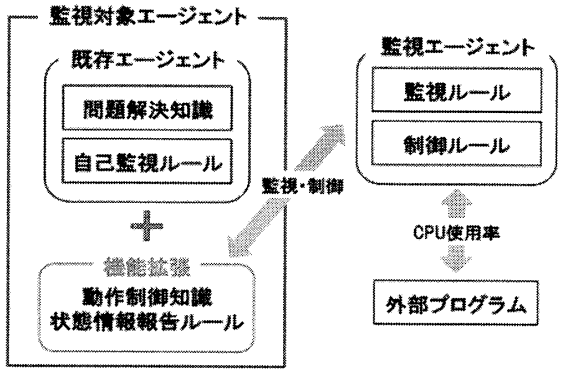

図 3：監視アーキテクチャの全体構成

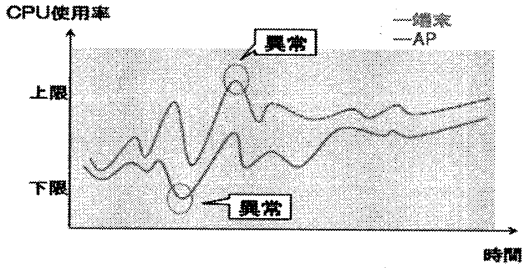

図 4：CPU 使用率による異常検出

端末全体の CPU 使用率及びエージェントプラットフォームの CPU 使用率のどちらか一方，あるいは両方が正常状態の範囲外であるとき，エージェントシステムに異常があると判断する(図 4)．

また，異常状態にあるとき，どのエージェントに異常が発生しているのか判断するため，収集･蓄積されているメッセージ到着間隔を用いる．

[3] 制御手法の選択

特定された異常の種類により，適切な再構成プランを決定する．プラットフォームの CPU 使用率が下限を下回っている場合は，エージェントが十分に機能していないことが考えられるため，当該エージェント特定後そのエージェントを再起動させる．端末全体の CPU 使用率が上限より高い場合は，エージェントの移動またはエージェント組織の再構成を行う．エージェントがタスク処理中であり，再構成によりそのタスクが中断され途中までの実行結果が破棄される等の不利益を被る場合は，エージェントを他の環境に移動させることで負荷を分散させる等の処理を行う．それらの問題がない場合は再構成を行い，異常エージェントを交換する．

[4] 代替エージェントの記憶

前章で説明したように，再構成時はタスク通知･入札･落札という三つのプロセスを経ることになる．このプロセスには入札待ち時間等が発生し，全体として迅速に再構成が行われない場合がある．そこで，初回の組織構成の入札時に入札をかけたが落札に漏れたエージェントを，再構成時に交換するエージェントの候補として記憶しておくことで，再構成時に契約プロトコルを使用することなく直接落札を行うことができ，より迅速な再構成が可能となる．

4. プロトタイプの試作と実験

4.1 実装環境

監視アーキテクチャのプロトタイプ実装に用いた開発環境を以下に示す．

OS： Windows Vista Home Basic
プログラム開発環境： Visual Studio 2008，NetBeans IDE 6.5.1
プログラム言語： Visual C++，Java2 J2SDK 1.6.0_13
C コンパイラ： Borland C++ コンパイラ 5.5
エージェントプラットフォーム： DASH(DASH1.9.7h)

4.2 監視アーキテクチャの実装

監視アーキテクチャのプロトタイプの実装内容を示す．

● **監視対象エージェントの拡張機能**
- 動作制御知識の自動付与は静的に記述
- 5 秒毎に監視エージェントに空のメッセージ送信
- 監視エージェントからの命令を受け停止

● **監視エージェント**
- 監視対象エージェントからのメッセージ収集
- メッセージ到着間隔の蓄積
- CPU 使用率の観測
- 異常検出･特定
- 異常エージェントへの停止命令

4.3 評価実験の構成

評価実験を行う上で，監視アーキテクチャを適用する対象として，模擬的にエージェントシステムを構成した．以下，評価実験において利用した各エージェントについて説明する．

Wa:Watch

監視アーキテクチャの中心部となる監視エージェント．前述のような機能を有する．

Ma:Manager

後述する HeavyAgent，LightAgent のマネージャエージェント．それら二つのエージェントと組織を構成するためのルールを持つ．

He:HeavyAgent

ハノイの塔を解くエージェント．任意の時点で処理を重くすることが可能である．

Li:LightAgent

Manager と組織を構成するエージェント．特に処理は行わない．

Om:OnsenManager

後述する OnsenIF と Aomori のマネージャエージェント．各エージェント間のメッセージ変換を行う．

Oi:OnsenIF

利用者からの検索要求を OnsenManager に送信するエージェント．

Ao:Aomori

青森県の温泉データを保持するエージェント．

Cn:CnpManager

組織構成要求を組織構成プロトコルに変換し，対象のエージェントに送信するエージェント．リポジトリ内で動作する．

ここで，評価実験の基本的な流れを説明する．最初に，監視対象となるエージェント組織を構成する．次に，監視エージェント Watch を起動し，エージェント組織の監視を行う．観測中のある時点で HeavyAgent の処理を重くすることで疑似的にエージェント組織に問題が発生したものとして動作制御を行う．問題発生の前後の端末全体とエージェントプラットフォームの CPU 使用率の変化，各エージェントからのメッセージ到着間隔のずれを観測し，監視エージェントによる監視が正しく行えたかを評価する．

4.4 評価実験

4.4.1 実験目的

実験の目的は以下の通りである．
- 監視エージェント自体のエージェントシステムに対する負荷の確認
- CPU 使用率観測による異常状態の検出の可否
- メッセージ到着間隔と異常エージェントとの関連性の観測
- 異常エージェントの停止が適切に行われるか

4.4.2 評価実験環境

評価実験に用いた実行環境を表 1 に示す．

表 1：実行環境

オペレーティングシステム	Windows Vista Home Basic
プロセッサ	Intel Celeron 2.00GHz
メモリ	2.00GB
システムの種類	32bit
エージェント プラットフォーム	DASH(DASH1.9.7h)

実験 1：監視エージェントの負荷

使用エージェント：Watch

ワークプレース内で Watch エージェントのみを起動させ，端末全体とエージェントプラットフォーム(AP)の CPU 使用率(単位：%)を観測したものを表 2 に示す．

表 2：観測データ（実験 1）

端末	10.50	14.23	9.57	8.33	18.62	16.43
AP	1.25	0.0	0.0	0.0	0.31	1.25

表 2 からエージェントプラットフォームの CPU 負荷が十分小さいことが分かる．ゆえに，監視エージェント自体の負荷は十分小さく，システムに大きな影響を及ぼさないことが分かる．

実験 2：異常エージェントの停止

使用エージェント:Watch，HeavyAgent

ワークプレース内に HeavyAgent を生成し，その後 Watch を起動し監視を行う．一定時間経過後 HeavyAgent に異常を発生させ，その観測を行う．観測されたメッセージ到着間隔のずれ(単位：msec)を表 3 上段に，CPU 使用率を表 3 下段に示す．

表 3：観測データ(実験 2)

He	11	10	12	1164	3639

端末	9.24	3.34	3.03	58.84	100.0
AP	3.43	2.50	1.56	51.78	90.77

表 3 から HeavyAgent に異常の発生した 4 回目の観測時に CPU 使用率が上がり，同時にメッセージ到着時間に明らかな遅延が発生していることが分かる．このことから，CPU 使用率とメッセージ到着間隔には何らかの関連性があることが窺える．

実験 3：異常エージェントの特定

使用エージェント:Watch，Manager，HeavyAgent，LightAgent，OnsenManager，OnsenIF，Aomori，CnpManager

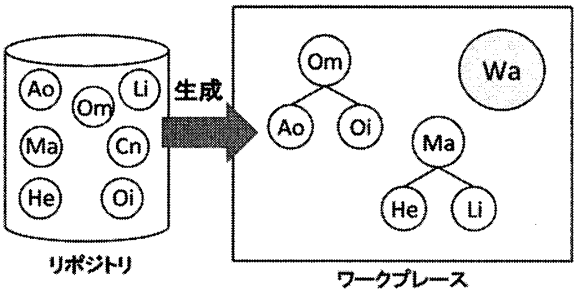

図 5：実験環境(実験 3)

図 5 に示すように，HeavyAgent，LightAgent，Manager からなるエージェント組織と，OnsenManager，OnsenIF，Aomori からなるエージェント組織を CnpManager を利用してリポジトリからワークプレース内に生成する．その後，Watch を起動し監視を行う．複数のエージェント組織が稼働している中での，CPU 使用率の変化や，各エージェントのメッセージ到着間隔のずれ，また異常発生時の異常エージェントの特定が可能かを観測する．観測された各エージェントのメッセージ到着間隔のずれを表 4 上段，CPU 使用率を表 4 下段に示す．

表 4：観測データ(実験 3)

Om	-71	-9	-83	56	3321	42985
Oi	78	-11	63	-6	3552	43173
Ao	-214	-5	-231	58	3224	42945
Ma	-93	-12	-88	-16	-44	43068
Li	68	-6	54	55	3956	43018
He	203	-7	188	16	45891	

端末	14.87	11.12	10.54	50.1	88.72	100.0
AP	11.23	9.36	8.42	9.05	68.89	93.91

表 4 から，異常が発生した時点で，問題のあるエージェント(HeavyAgent)のメッセージ到着間隔のずれが最大となっていることが分かる．しかし，Aomori などにも起動当初から他のエージェントよりも大きなずれが見られる等のことから，メッセージ到着間隔はエージェントの特性にも左右されることが考えられる．

4.4.3 考察

実験の結果から，各機能の有効性について考察する．

- **状態情報の収集**

  各エージェントに定期的にメッセージを送らせる機能については，一定の有効性と発展性がみられる．各エージェントから状態情報を送らせることで，監視エージェントによりシステム全体の把握が可能になったと言える．また，今回は実装していないが，CPU 使用率に異常が出ていない状態で，メッセージ到着間隔にずれが見られる，メッセージが届かないと言った場合に，異常エージェントがすぐに特定できる等といった方法も考えられる．

- **異常検出**

  今回の実験では，メッセージ到着間隔のずれから異常エージェントの特定を行っていた．各実験の結果から，メッセージ到着間隔に影響を及ぼすものに，CPU 使用率とエージェント特性が考えられる．

  まず，CPU 使用率によりメッセージ到着間隔にずれが発生するのは明らかである．CPU 負荷が増大すると，各実験で分かる通りメッセージ到着間隔のずれが確認された．これは計算機環境にも依存すると思われ，低スペックな CPU であるほど，このずれは大きなものになると考えられる．そして，各エージェントの持つ機能や知識といったエージェント特性によっても，メッセージ到着間隔には差が生じた．

以上のことから，本稿で提案した監視アーキテクチャの有効性について考えると，今回実装したプロトタイプの様に，メッセージ到着間隔のずれのみで異常エージェントを特定するのは，信頼性に欠けると考えられる．その理由として，メッセージ到着間隔は，周りの環境によってかなり左右され安定せず，複数のエージェントに異常が発生した場合は特定が困難といったことが挙げられる．しかし，メッセージ到着間隔のずれと異常エージェントには一定の関連性が見受けられるため，これを監視パラメータの一つとして扱い，他のパラメータと併用することは可能と考えられる．

5. おわりに

本研究の目的は，従来のエージェントシステムにおける，知識の局所性や静的な動作知識等の問題を解決し，安定性の高いエージェントシステムを実現することである．そこで本稿では，まず従来型のエージェントシステムにおける問題点をあげ，その解決案として提案されている発展型エージェントシステムに着目し，その実現法として監視アーキテクチャを提案した．監視アーキテクチャは，各エージェントに監視用の制御知識を加え状態情報を収集することで，システム全体を考慮した動作制御を可能とし，より安定性の高いエージェントシステムの実現を可能とする．

また，監視アーキテクチャのプロトタイプを設計･実装し，実際に稼働させ実験･評価を行った．限定的な場面ではあるが，提案手法によるシステム全体の監視が行えることを確認し，この手法による発展型エージェントシステム実現の可能性について示した．

今後の課題として，他のパラメータを組み込むことによる信頼性の向上や，監視ルールの自動付加の実装，再構成プランの選択，他環境を視野に入れた実装などが挙げられる．

参考文献

[1] 打矢隆弘，前村貴秀，今野将，木下哲男，“発展型システムのための高機能エージェントリポジトリ”，合同エージェントワークショップ＆シンポジウム JAWS2006 講演論文集，2006.

[2] 打矢隆弘，前村貴秀，木下哲男，“発展型ソーシャルウェアのためのリポジトリ型フレームワーク（Ⅲ）”，信学技報 AI2008-24(2008-11), pp. 1-6, 2008.

[3] 前村貴秀，石井貴光，打矢隆弘，今野将，木下哲男，“発展型エージェントシステムのための柔軟な組織再構成方式”，合同エージェントワークショップ＆シンポジウム JAWS2007 論文講演集，2007.