マルチエージェント系の協同開発のためのマルチエージェント系

A multiple agent system for cooperative development of multiple agent systems

鷹合 基行　武田 英明　西田 豊明

Motoyuki Takaai　Hideaki Takeda　Toyoaki Nishida

奈良先端科学技術大学院大学 情報科学研究科

Graduate School of Information Science, Nara Institute of Science and Technology

In this paper, we propose a new environment that supports cooperative development of multiple agent systems (MAS). Since it is natural that MAS is developed cooperatively by multiple programmers, it is important to provide environments that support their cooperative work. For this purpose, we suggest a system called *GuardNet*. In *GuardNet*,which is also realized as MAS, there exists an agent called *Guardant* for every agent in target MAS. *Guardants* can store requirement specifications of their for target and relevant agents, and exchange them with each other on behalf of agent programmers. Another feature of *GuardNet* is *agent-imitating* facilities for debugging and testing. In *agent-imitating*, a *Guardant* behaves as a proxy either for the target agent or relevant agents. The proxy simulates the agent based on the requirement a specifications determined by *Guardants'* communication. Using *agent-imitating* facilities, we can test the agent system which includes incomplete agents.

1　はじめに

近年マルチエージェント系による問題解決や、知識情報処理についての研究が盛んに行なわれている。マルチエージェント系とは、いくつかのエージェントと呼ばれる情報処理システムを連結しそれらの相互作用によってより大きな問題を解決していくものである。我々は知識コミュニティと呼ぶ、マルチエージェント系による大規模知識ベースの実現にむけての知識の共有と再利用の枠組を提唱し、システムの開発を行ってきた[4]。

マルチエージェント系は複数の開発者によって開発されることが多いが、開発者同士の協調が十分なされていないとエージェント同士の相互作用がうまく開発できない。また、エージェントは他のエージェントが存在して初めて機能するので、エージェント単体での動作チェックが困難である。

そこで、本研究ではエージェント開発者同士の協同開発を支援する環境をマルチエージェント系で構成した GuardNet と呼ぶシステムを提案する。 GuardNet ではエージェントが開発者に代わって開発中のエージェントの仕様を交換する。また、そこで得られた情報をもとに、エージェント系の動作のシミュレーションを実現する。

2　GuardNet による開発者同士の協同開発の支援

GuardNet は図 1のようにエージェント開発者ごとに開発環境エージェント (Guardant と呼ぶ) を配置し、それらのメッセージ交換によってエージェントの協同開発を支援する。

マルチエージェント系の開発環境には柔軟な環境が必要である。よって、 GuardNet では開発者同士の協同開発を支援する環境をマルチエージェント系によって実現した。その結果、エージェントの開発者はエージェントの仕様の決定とその仕様にもとづく実行シミュレーションが同一の開発環境で行えるようになった。

GuardNet のエージェント系は、開発対象のエージェント系とは別に構成されている。開発におけるメッセージが開発対象のエージェント系に混入し、混乱しないようにするためである。

GuardNet は Guardant と呼ばれるエージェントから構成される。 Guardant は開発対象のエージェントごとに配置される開発環境である。すなわち、一つの開発対象エージェントにつき (開発者, 開発エージェント,Guardant) の組が開発の単位となる。 Guardant の機能は以下に挙げられる。

- 他の Guardant との通信によるメッセージの仕様の決定
- 虚像化と呼ぶエージェントの動作シミュレーション
- エージェントの仕様に基づくエージェントプログラミングの支援

図 2に Guardant の構成を示す。 Guardant は GuardNet を構成するエージェントであると同時に、開発対象のエージェント系のエージェントでもある。

仕様交換マネージャは GuardNet のエージェント系を用いてエージェント開発者同士の協調によるメッセージ仕様の決定を支援し、メッセージ仕様を仕様データベースに登録する。外部および開発対象のエージェントのシミュレータは、それぞれ外部および開発対象の仕様データベースをもとに開発対象のエージェント系に対し虚像化を実現する。

連絡先: 奈良先端科学技術大学院大学 情報科学研究科, 〒 630-01 生駒市高山町 8916-5, 電話:07437-9-9211 内線 5316, fax:07437-2-5269,
E-mail: motoyu-t@is.aist-nara.ac.jp,

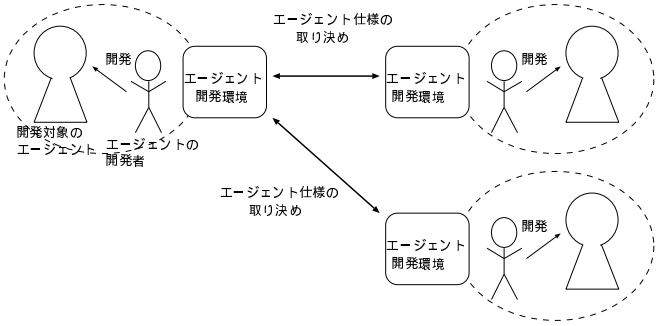

図 1: **GuardNet** による開発者同士の結び付け

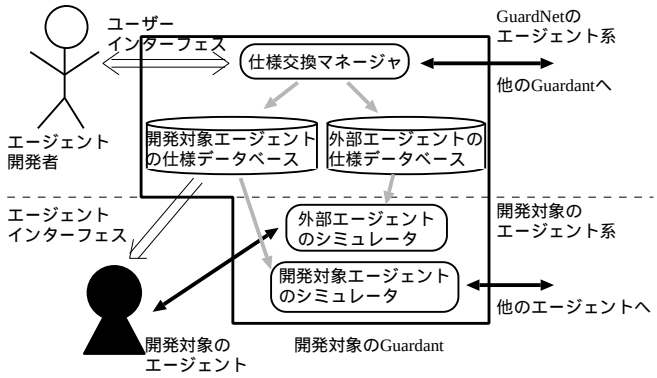

図 2: Guardant

3 Guardant によるメッセージ仕様の決定

エージェントは何らかの知識をメッセージ交換を通して提供するものと考えられるので、エージェントの仕様とはメッセージの仕様の集合であるといえる。

GuardNet は複数のエージェント開発者によるメッセージの仕様の決定のための協同作業を支援する。

メッセージの通信プロトコルとして KQML[1]、知識表現言語として KIF[2] を用いた場合のメッセージ仕様の例を図 3に挙げる。ここで、寺の情報を持つエージェントがこのメッセージ仕様を満たすとき、 $name に具体的な寺の名前が代入されたメッセージに対して ? で始まる変数に対応する具体的な情報、つまり寺の拝観時間を返すことが期待される。

メッセージ仕様決定の過程

開発者 U がエージェント A の Guardant G を用いて他のエージェント開発者 U' (エージェント:A',Guardant:G') にメッセージの仕様を要求している場面を考える。そのときの GuardNet における仕様決定の流れは以下のようになる。

1. U が要求するメッセージ仕様の概要を G に登録
2. メッセージ仕様の取り決め

```
(ask-if
   :content (and (temple     ?x $name)
                 (open-time  ?x ?o-time)
                 (close-time ?x ?c-time))
   :language kif)
```

図 3: メッセージ仕様の例

   (a) U から G へ要求するメッセージ仕様を入力
   (b) G から G' へ要求したメッセージ仕様の伝達
   (c) G' から U' へ要求したメッセージ仕様の提示
   (d) U' から G' へ要求したメッセージ仕様に対する返答 (受諾 or 反論) を入力
   (e) G' から G へ返答の伝達
   (f) G から U へ返答の提示
3. G, G' の仕様データベースに要求中または決定済の仕様を記録

ここで、 (a) から (f) の過程は会話の状態遷移モデル [6] に基づいて Guardant 内の仕様交換マネージャを用いて実行され、双方が受諾するまで繰り返される。またこの過程中, 要求中あるいは決定済の仕様は G,G' の仕様データベースに随時記録される。状態遷移は登録されたメッセージ仕様ごとに用意されるので、ユーザーは複数のメッセージ仕様決定の過程を並行して行うことができる。

外部エージェントに要求したメッセージ仕様は外部エージェントの仕様データベースに蓄えられる。逆に外部から仕様要求された場合、その要求されたメッセージ仕様は開発対象エージェントのデータベースに置かれる。

4 エージェントの虚像化

エージェントの虚像化とは、存在しないあるいは不完全なエージェントを他のエージェントがあたかもそのエージェントであるかのように振舞わせることである。この方法を用いることで存在しないあるいは不完全なエージェントを含むエージェント系の実行シミュレーションが可能になる。ここでは 3 節で述べたメッセージ交換で得られる仕様をもとに虚像化を実現する。

我々は次の二つのタイプの虚像化を行う。

外部エージェントの虚像化 開発されたエージェントを単独に動作チェックする目的で、そのエージェントが依存する外部エージェントを虚像化する。

開発対象エージェントの虚像化 トラブルによるエージェントの欠損に対する応急処置やエージェント系全体の動作チェックを行う目的で、動作していない開発対象のエージェントを虚像化する。

4.1 外部エージェントの虚像化

開発対象のエージェントが他のエージェントの知識に依存している場合、図 4のように開発の済んでいないエージェントがあると開発対象のエージェントの動作チェックができない。依存関係のある開発の済んでいないエージェントを虚像化により実行シミュレーションすることによって開発対象のエージェントの動作チェックを可能にする。

対象エージェントからみた外部のエージェントのメッセージ行為は次の四つに分類される。

質問 外部へ質問メッセージを送り返答が返ってくるのを期待する。

返答 外部からの質問メッセージに対し返答メッセージを送り返す。

情報発信 メッセージ送信によって外部に対象エージェントの情報を伝える。

情報受信 外部からのメッセージによって情報を受け取る。

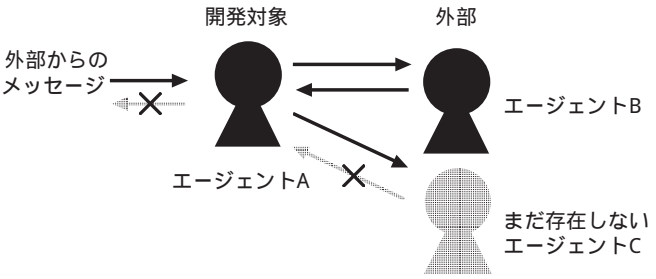

図 4: エージェントの依存関係

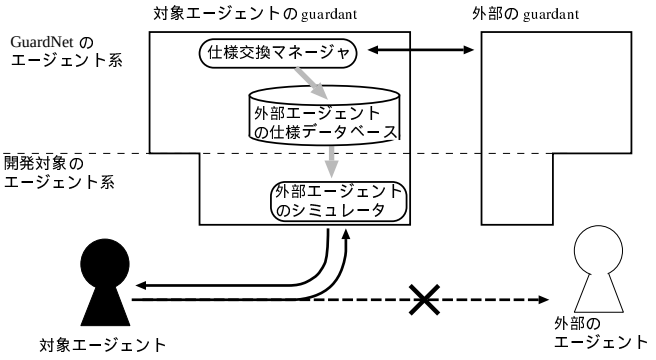

図 5: Guardant による外部エージェントの虚像化

エージェントの実行シミュレータはこれら四つのメッセージ行為をシミュレーションしなければならない。特に、返答は他の三つと違い質問に対応する返答メッセージを送らねばならず、実行シミュレーションも複雑である。以下では、返答のシミュレーションについて述べる。

虚像化は Guardant の持つメッセージ仕様をもとに実現される。メッセージ仕様データベースは質問メッセージに対する具体的な返答の例も記述できるので、エージェント開発者自身が質問に対する返答の例を付加しておき、このデータを用いて質問に対する応答を検索して返答のシミュレーションを行う (図 5参照)。

メッセージフローの制御

虚像化を行うためにはエージェントの実行シミュレータとメッセージが交換ができるようにメッセージフローを制御する必要がある。

外部エージェントが虚像化されている場合、開発対象エージェントからその外部エージェントへ送られるメッセージの全ては対応する Guardant の外部エージェントシミュレータによって受信され、実際の外部エージェントが受け取ることはない。外部エージェントシミュレータからの返答には、差出人 (sender) を虚像化されている外部エージェントとしたメッセージを用いることで、実際の外部から出されたメッセージを装うことができる。

連邦アーキテクチャ [5] を採用しているエージェント系では、全てのメッセージは一旦 facilitator と呼ばれる特殊なエージェントに送られる [3]。そこで、このようなメッセージフローの制御は facilitator を用いて実現される。

4.2 開発対象エージェントの虚像化

開発対象のエージェントが欠けているとエージェント系全体の実行が不可能となる。開発の済んでいない開発対象エージェントを虚像化により実行シミュレーションすることによってエージェント系全体の実行を可能にする。

表 1: 試作した Guardant

	コミュニケーション相手	機能
Guardant	GuardNet のエージェント系, 開発対象のエージェント系	エージェント仕様の管理・虚像化機能。
GuardantUI	GuardNet のエージェント系, ユーザー (人間)	Guardant のユーザーインタフェス

外部エージェントの虚像化と同様に実行シミュレーションしなければならないメッセージ行為は、質問・返答・情報発信・情報受信の四つが挙げらる。シミュレーションの方法も同様に、 Guardant の持つメッセージ仕様データベースの情報をもとに行われる。

メッセージフローの制御

開発対象エージェントが虚像化されている場合、開発対象エージェントへ送られる全てのメッセージは、その Guardant が持つ開発対象エージェントシミュレータによって受信される。もし開発対象エージェントが存在していてもそれが受け取ることはない。開発対象エージェントシミュレータからのメッセージは、外部エージェントの虚像化の時と同様に差出人 (sender) の書き換えによって開発対象エージェントからのメッセージであるかのように装ってメッセージが送られる。

5 Guardant によるエージェントプログラミングの支援

Guardant はエージェントインターフェスを通してエージェントプログラミングを支援することができる。 Guardant には開発対象のエージェントのメッセージ仕様が格納されているので、それに基づいてエージェントのテンプレートを作ることができる。例えば、メッセージ駆動型エージェントのためのメッセージ登録テンプレートを生成することができる。

6 知識コミュニティにおける実装

我々は GuardNet を知識コミュニティにおいて試作した。

試作した GuardNet システムは以上に述べた機能のうちエージェントのメッセージ仕様決定支援とエージェントの外部エージェントの虚像化を実装しているが、開発対象エージェントの虚像化とエージェント作成支援はまだ未実装である。

Guardant は実際には 2 つのエージェントから構成される (表 1)。

Guardant は GNU Common Lisp を用いて実現されている。 Guardant はユーザーが開発環境を使用していない時でも動き続け、他の Guardant からのメッセージなどに備える。メッセージ仕様の状態やデータなどはすべてこの Guardant が保持し、 GuardantUI が持つことはない。

GuardantUI は Emacs Lisp (ver 19) によって実現されている。ユーザーは好きな時にいつでも GuardantUI を立ち上げ・終了が可能である。

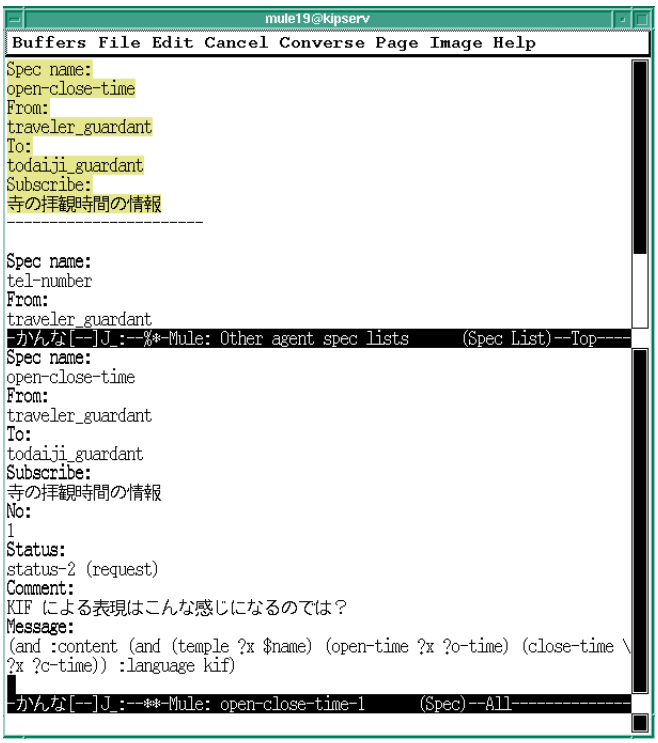

図 6: 旅行者エージェントの開発者が Guardant を用いて仕様交換している様子

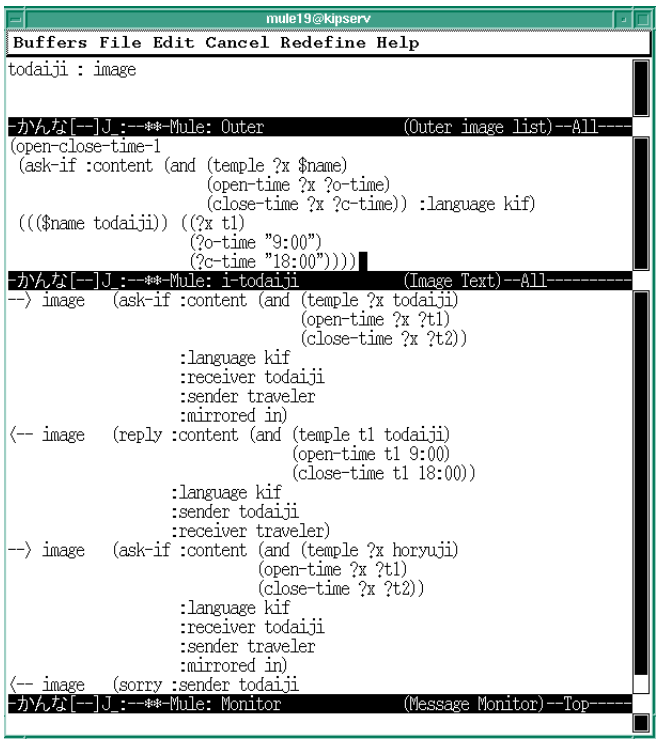

図 7: 旅行者エージェントの開発者が Guardant を用いて仕様交換している様子

試作システムの実行例

図 6は、試作システムにおいて旅行者エージェントの開発者が Guardant を用いて東大寺エージェントの開発者に拝観時間についての仕様を要求している様子である。上のウィンドウは現在登録されているメッセージ仕様が表示されている。その中の寺の拝観時間の情報に関する仕様が選択 (ハイライトの部分) されており、そのメッセージ仕様についての仕様交換の内容が、下のウィンドウに表示されている。ここでは、旅行者エージェントの開発者が東大寺エージェントの開発者に、図 3のメッセージ仕様を要求 (request) している。要求した内容は東大寺エージェントの Guardant のほうに送られ、その後東大寺エージェントの Guardant から回答を受け取ることになる。

図 7は、試作システムにおいて旅行者エージェントの開発者が東大寺エージェントの開発者に拝観時間についての仕様を要求している様子である。上のウィンドウは東大寺エージェントが現在虚像化中であることを示している。中央のウィンドウには、東大寺エージェントの図 3のメッセージ仕様のデータが表示されている。エージェント開発者はここでシミュレーションのための返答の例として、東大寺の拝観時間 (9 時から 18 時まで) の情報を付加している。下のウィンドウは虚像化による東大寺エージェントの実行シミュレーションの様子である。旅行者エージェントから東大寺エージェントへ出されたメッセージを、東大寺エージェントに代わって旅行者エージェントの Guardant が受け取り、返答を行っている。

7 結論と展望

本研究ではエージェントの協同開発を支援するマルチエージェント系 GuardNet について述べた。本システムを用いるとエージェント開発者同士が離れていてもエージェント開発が可能になり、エージェントのデバッグも効率良く行える。

参考文献

[1] T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, J. McGuire, P. Pelavin, S. Shapiro, and C. Beck. Specification of the KQML agent-communication language. Technical Report EIT TR 92-04, Enterprise Integration Technologies, 1992. (Updated July 1993).

[2] Michael R. Genesereth and Richard Fikes. Knowledge Interchange Format version 3.0 reference manual. Technical Report Logic-90-4, Computer Science Department, Stanford University, 1990.

[3] 飯野健二, 武田英明, 西田豊明. 知識コミュニティにおけるメッセージ仲介機構. 人工知能学会全国大会 (第 8 回) 論文集, pp. 287–290, 1994.

[4] 西田豊明. 知識コミュニティ. 人工知能学会全国大会 (第 8 回) 論文集, pp. 77–80, 1994.

[5] R. S. Patil, R. E. Fikes, P. F. Patel-Schneider, D. McKay, T. Finin, T. R. Gruber, and R. Neches. The DARPA knowledge sharing effort: Progress report. In Charles Rich, Bernhard Nebel, and William Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*. Morgan Kaufmann, 1992.

[6] Terry Winograd. A language / action perspective on the design of cooperative work. In Irene Greif, editor, *Computer-Supported Cooperative Work: A Books of Reedings*. Morgan Kaufmann, 1988.