プログラミング 2−20
（1995. 8. 25）

分散環境下におけるマルチエージェントシステム記述用言語

浦田 泰裕，齋田 明生
神戸大学自然科学研究科

田村 直之，金田 悠紀夫
神戸大学工学部

川村 尚生
鳥取大学工学部

本稿では，Prologを基にしたマルチエージェントシステム記述用言語を提案する．このシステムでは，エージェントに知識ベースを付与する機能を持った“フィールド”と呼ぶ機構を導入する．フィールドはPrologプログラムの形で知識ベースを保持し，エージェントはフィールドに所属することによって，フィールド内の知識を共有できる．フィールドに所属するエージェントはフィールドを介してコミュニケーションを行なう．本研究のマルチエージェントシステム記述用言語の処理系のプロトタイプがネットワーク上のワークステーションで実装されている．

Multi-agent Programming System on a Workstation Network

Yasuhiro URATA,Akio SAITA,
The Graduate School of Science and Technology, Kobe University
Naoyuki TAMURA and Yukio KANEDA,
Faculty of Engineering, Kobe University
Takao KAWAMURA
Faculty of Engineering, Tottori University

In this paper we propose a programming system for developing multi-agent systems based on Prolog. In this system, we introduced “field” as a knowledge base for agents. Fields can contain a knowledge base in a form of Prolog program. Agents can enter a field and share the knowledge in the field. Agents in a field can communicate each other through the field. A prototype programming system based on this model was built on a workstation network.

1 はじめに

マルチエージェントシステムは，人工知能を含む多くのソフトウェアシステムのモデルとして注目されている．マルチエージェントシステムでは，多くの自律的なエージェントが相互に協調して仕事をする．

現在，様々なマルチエージェントシステム記述用言語が提案されており，場を介した通信を行なうLinda，グループ化のために場を導入したKamui88，エージェント自体が階層構造を持つCellula，Kemari，エージェント間の知識コミュニケーションをサポートするための構造を持つMOAP等がある．エージェントはプロセスまたはタスクとして生成され，それぞれの環境において様々な仕事をする．エージェントが新しい環境，仕事をする場合，動的に新しいデータベースや手続きを保持する必要がある．

そこで，多くのエージェントが協調している場合，エージェント間の共有な知識ベースが必要とされる．この研究では，“フィールド”と呼ばれる知識の共有のためのオブジェクトを提案する．エージェントはフィールドに入ることによってフィールドが所有する共有知識を参照できるようになる．同じフィールドに所属するエージェントは知識を共有する．本研究で提案するPrologを基にしたマルチエージェントシステム記述用言語はエージェントの生成，制御，フィールドの生成，制御，エージェント間の通信機能を提供している．さらに，分散されたネットワーク環境におけるマルチエージェントシステム間の協調を実現している．

2 マルチエージェントシステム記述用言語

本研究で提案するマルチエージェントシステム記述用言語は，Prologを基にしており，エージェント，フィールドを実現する機構をもつ．

多くのエージェントがシステム内に存在し，各エージェントは自律的に行動する．フィールドは所属するエージェントをグループ化するだけでなく，エージェントに共有な知識ベースを与える機能をもつ．同じフィールドに属するエージェントは1つのグループを形成する．エージェント間の通信は主にフィールドの知識ベースを介して行なわれるが，同じフィールドに属さないエージェントとは1対1の通信を行なう．

2.1 エージェント

エージェントは以下の機能を持つ．

(1) エージェントの行動が定義されたプログラムを実行する．
(2) エージェントが属するフィールドに含まれるプログラムを実行する．
(3) 他のエージェントとの通信を行なう．
(4) dynamic agentの生成，消去を行なう．
(5) フィールドに対する操作を行なう．

エージェントにはstatic agentとdynamic agentが存在する．前者はstart_up agentによって生成され，システムがシャットダウンされるまで存在する．一方後者は他のエージェントが組み込み述語create/2を実行すると生成され，組み込み述語destroy/1を実行すると消去される．

2.2 エージェントの記述と生成

エージェントの行動を記述する定義プログラムは，以下の宣言を用いて与えられる．

```
def_agent agent_class
   main(Self):- /* 初期手続き */
      /* その他の手続き */
enddef.
```

main/1の手続きはエージェントが生成された時に実行される．static agentは次の組み込み述語によってstart_up agentにより1つだけ生成される．agent_nameがエージェントに固有の名前となる．

```
s_create(agent_class, agent_name).
```

dynamic agentは次の組み込み述語によって他のエージェントにより生成される．

```
create(agent_class, Agent_ID).
```

agent_classで与えられるエージェントが生成され，引数Agent_IDにはシステム内でエージェントに固有のIDが代入される．次の組み込み述語によってdynamic agentは消去される．

```
destroy(Agent_ID).
```

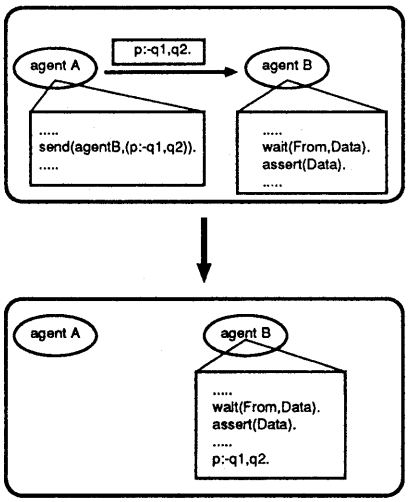

図 1: エージェントの知識の動的変化

Prolog にはプログラムとデータの区別がないので，エージェントのプログラムは生成時に与えられるだけでなく，実行中に追加，削除が可能である．例えば，図 1 では，節'p:-q1,q2.'はメッセージとして送信され，受信したエージェントのプログラムに追加される．

2.3 フィールドの記述と生成

フィールドの共有知識を記述する定義プログラムは，以下の宣言を用いて与えられる．

```
def_field field_class
    共有知識
enddef.
```

static field は次の組み込み述語によって start_up agent により 1 つだけ生成される．field_name がフィールドに固有の名前となる．

```
s_create_field(field_class,
                        field_name).
```

dynamic field は次の 2 つの組み込み述語によってエージェントにより生成，消去される．

```
create_field(field_class, Field_ID).
destroy_field(Field_ID).
```

共有知識はフィールド内に蓄えられる．共有知識とエージェントプログラムの表現方法は同じである．エージェントがフィールドに入った時，そのフィールドの共有知識にアクセスできる（図 2）．

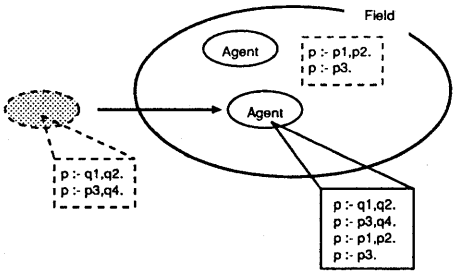

図 2: エージェントは自身の共有知識の中にフィールドの共有知識を取り込む

エージェントは自身とフィールドの知識を合わせて利用できる．そのため，エージェントは自身で全ての知識を持つ必要はなく，必要な情報を提供する適切なフィールドに所属して問題を解決する．図 3 のようにフィールドに属するエージェントは自身の知識をフィールドの共有知識をまとめた一つの知識ベースとしてアクセスできる．知識ベースの検索の順番は，まずエージェント自身の知識が検索され，その後，フィールドの共有知識が検索される．エージェントは複数のフィールドに所属できるので，この場合は，フィールドに入った順番と逆，つまり，最後に入ったフィールドの共有知識から検索される．

2.4 フィールドに対する操作

エージェントのフィールドへの出入りは次の組み込み述語で行なう．

```
enter_field(Field).
exit_field(Field).
```

enter_field/1,exit_field/1 はバックトラックによってキャンセルされる．図 3 は 3 つのフィールドが用意されていて，エージェントがそれらのフィールドに順番に入って処理を行なう例である．test_X に失敗したら，バックトラックが起こりエージェントは自動的にフィールドから出て，次のフィールドに入る．

```
(enter_field(Field_a),test_a;
 enter_field(Field_b),test_b;
 enter_field(Field_c),test_c).
```

図 3: enter_field/1 はバックトラックによってキャンセルされる.

同じフィールドに多くのエージェントが所属する場合について考える. フィールド内の知識はエージェントによって共有されている. そのため, フィールド内の知識を変更することによって他のエージェントに影響を与えることができる.

次の組み込み述語でフィールドの共有知識を変更することができる.

```
share(Field, Assert, Retract).
```

Field でフィールドを指定し, Assert,Retract は節のリストでそれぞれ追加と削除される節を指定する.

2.5 メッセージ通信

エージェントは他のエージェントまたはフィールドにメッセージを送信することができる. メッセージの送信は次の組み込み述語によって行なわれる.

```
send(To,Message).
```

To によって送信先のエージェントまたはフィールドを指定する. Message によって送信するメッセージを指定する. 送信先がエージェントであれば通信は 1 対 1 であるが, 送信先がフィールドであればメッセージはフィールドに属するエージェント全てに送られる. 送信は非同期的に行なわれ, send/2 はただちに終了する.

メッセージの受信は次の組み込み述語によって行なわれる.

```
receive(From, Message).
wait(From, Message).
```

receive/2 は From,Message ともにユニファイ可能ならば成功するが, ユニファイ可能なメッセージが届いていなければ失敗する.

wait/2 は From,Message ともにユニファイ可能なメッセージが届くまで待つ.

3 実装

本研究におけるマルチエージェントシステム記述用言語の実装は Sun ワークステーション上のアプリケーション PVM(Parallel Virtual Machines)を用いて行なった. PVM ではタスクの動的生成が可能なためエージェント, フィールドの実装が容易である. 実行のながれは, 最初にマスタータスクが生成されエージェント, フィールドが宣言されているファイルを読み込み, それぞれ生成する. マスタータスクがエージェント間のメッセージ通信を仲介するので, static agent,static field に対して agent_name, field_name を用いての通信が可能である.

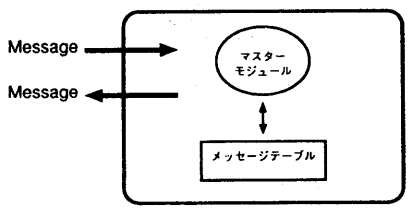

図 4: マスタータスクの構成

各エージェントは Prolog インタプリタを含むタスクとして実現され, 宣言によって与えられたプログラムを実行する. エージェントは以下のもので構成されている.

(1) Prolog モジュール
(2) Prolog プログラム
(3) メッセージプール

Prolog モジュールはエージェントの宣言によって与えられるプログラムを実行する Prolog インタプリタである.

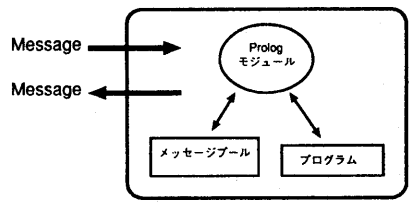

図 5: エージェントタスクの構成

他のエージェントからのメッセージはメッセージプールに蓄えられる．

各フィールドにもタスクを割り当て，エージェントからの要求を処理する．エージェントからの要求には次のものがある．

(1) フィールド内のエージェントに対するブロードキャスト．
(2) フィールド内のあるエージェントに対する対象不特定通信．
(3) 共有知識の要求．

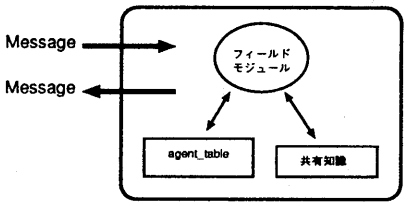

図 6: フィールドタスクの構成

実際にエージェントがフィールドの共有知識を必要とする場合は，1番最後に所属したフィールドに求める知識を要求し．そのフィールドの共有知識でも足りなければ，順に次のフィールドに求める知識を要求する．

share/3 によりフィールド内の共有知識が変更された場合は，その知識を与えたエージェントに対してその知識を削除するように要求する．エージェントが変更された知識が必要になれば，再びフィールドに要求するのでエージェントは変更された知識を得ることができる．

4 サンプルプログラム

本章では，本マルチエージェントシステム記述用言語を用いた応用例として，生産者-消費者問題の消費者を記述する．生産者-消費者問題とは生産者によって生産されたものを消費者が定められた処理を行なう問題である．

本マルチエージェントシステム記述用言語では以下の手順で実行される．

(1) start_up エージェントが，生産者エージェント，消費者エージェント，job フィールドを生成する．
(2) 生産者エージェントは job が発生すると，job フィールドに job を書き込み，消費エージェントにメッセージを送信する．
(3) 消費者エージェントは生産者エージェントからメッセージを受けとると，そのフィールドに入り job を取り出し，処理を行なう．

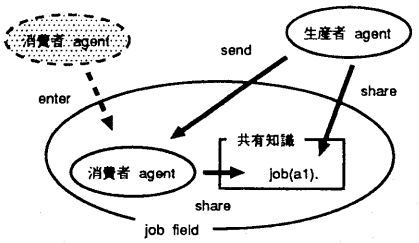

図 7: 生産者-消費者問題

この問題では，本記述用言語の特徴であるフィールドの共有知識とエージェント間の非同期通信を利用している．

```
%生産者-消費者問題における消費者の記述

def_agent start_up
   main(Self) :- s_create_field(job_field, j1),
                 s_create(seisan_agent, seisan_a),
                 s_create(syohi_agent, syohi_a).
enddef.

def_agent seisan_agent
   main(Self) :- seisan.

   seisan :- ......,
             share(j1, [job(a1)],[]),
             send(syohi_a, please),
             seisan.
enddef.

def_agent syohi_agent
   main(Self) :- enter(j1),
                 syohi.

   syohi :- wait(seisan_a, please),
            share(j1,[],[job(X)]),
            ......,syohi.
enddef.

def_field job_field

enddef.
```

5 ネットワーク環境への拡張

近年，多くのコンピューターシステムがお互いにネットワークで接続されている．複数のエージェントシステムがネットワーク上に存在し，

それらのエージェントシステム間での協調も考えることができる．この場合，２つの問題点がある．

(1) 他のエージェントシステムからの知識獲得方法．すなわち，どこにどのような知識が存在するのか．
(2) エージェントシステム間のメッセージパッシングの方法．

(1)の解決法としては，bulletin を作成して，他のエージェントシステムに存在する知識を登録する．他のエージェントシステムに質問をする場合，bulletin で求める知識を所有するエージェントシステムを調べて，メッセージを送る．(2)の解決法としては，すべてのエージェントシステムに name(ID)が与えられ，これを通信の際に利用する．メッセージは Prolog の節の形をとっていて，文字列で表現されていて，指定されたエージェントシステムが受信することができる．

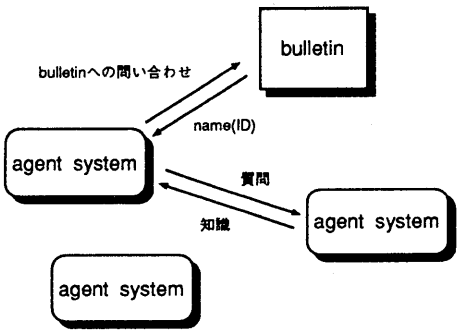

図 8: エージェントシステム間の協調

メッセージとして Prolog プログラムを送り，受けとったシステムがそれを実行することは可能である．それゆえ，エージェントの他エージェントシステムへのマイグレーション（移住）の実現が可能である．

6 結論

本研究におけるマルチエージェントシステム記述用言語では，共有知識を持ち，エージェントのグループをつくるフィールドと呼ばれる機構が提案された．このフィールドを用いて

(1) エージェントのグループ内の共有知識．
(2) エージェントの属する環境の表現．
(3) 分散したコンピューターネットワーク内のエージェントシステム間の協調．

が実現された．

このモデルに基づいて，Prolog を拡張したプログラミング言語が設計され，そのプロトタイプが分散したコンピューターネットワーク上で構築されている．その効果はサンプルプログラムによって確認されている．

参考文献

1) Rosenschein, J. S. and Zlotkin, G.: Designing Conventions for Automated Negotiation. AI MAGAZINE FALL, (1994), pp.29–46.
2) Tsichritzis, D., Fiume, E., Gibbs, S. and Nierstrasz, O.: KNOs : KNowledge Acquisition, Dissemination, and Manipulation Objects. ACM Transactions on Office Informaton Systems, Vol.5, No.1(1987),pp.96–112.
3) Carriero, N. and Galernter, D.: Linda in Context. Communications of the ACM,Vol.32,No.4,(1989),pp.444–458.
4) 吉田紀彦，楢崎修二：場と一体化したプロセスの概念に基づく並列協調処理モデル Cellula ．情報処理学会論文誌，Vol.31,No.7(1990),pp.1071–1079 .
5) 矢野博之，武宮博，布川博士，野口正一：自律的な協調処理を行なう分権型計算モデル Kemari ．情報処理学会論文誌，Vol.33,No.12(1992),pp.1476–1485 .
6) 渡辺慎哉，原田康徳，三谷和史，宮本衛市：場とイベントによる並列計算モデル（Kamui88）．コンピュータソフトウェア，Vol.6,No.1(1989),pp.41–55 .
7) 丸一威雄，市川正紀，所真理雄：自律的エージェントからなる組織の計算モデルと分散協調問題解決への応用．情報処理学会論文誌，Vol.31,No.1(1990),pp.1768–1779 .
8) 西尾郁彦，渡辺豊英，杉江昇：オブジェクトと場に基づいた協調的プログラム言語．情報処理学会論文誌， Vol.34, No.12(1993), pp.2499–2508 .
9) 中島秀之：有機的プログラミング言語 Gaea におけるエージェント間通信．電子情報

通信学会信学技報, AI94-42,(1994),pp.71–79.

10) Woo, C. C. and Lochovsky, F. H.: Knowledge Communication In Intelligent Information Systems. International Journal of Intelligent and Cooperative Information Systems, Vol.1, No.1(1991),pp.203–228.

11) R.E.フィルマン, D.P.フリードマン,(訳: 雨宮真人, 尾内理紀夫, 高橋直久): 協調型計算システム～分散型ソフトウェアの技法と道具立て～. マグロウヒルブック社.

12) Geist, A., Beguelin, A., Dongarra, J., Jiang, Weicheng., Manchek, R., Sunderam, V.: PVM3 USER'S GUIDE AND REFERENCE MANUAL. (1993). (邦訳: 「PVM3 ユーザーズガイド & リファレンスマニュアル」, 村田英明).