EAMMO 記述処理系によるサッカーエージェントの設計

非会員 伊藤 暢 浩 (名古屋工業大学 知能情報システム学科)
 非会員 中川 浩 一 (名古屋工業大学 知能情報システム学科)
 非会員 杜 小 勇 (名古屋工業大学 知能情報システム学科)
 正員 石井 直 宏 (名古屋工業大学 知能情報システム学科)

The Design of Soccer Agents based on EAMMO Description-Processing System

Nobuhiro Ito, Non-member, Kouichi Nakagawa, Non-member, Xiaoyong Du, Non-member, Naohiro Ishii, Member (Department of Intelligence and Computer Science, Nagoya Institute of Technology)

Some real-world entities such as human beings have different aspects in different situations (multiple aspects). They can decide one of their aspects dynamically and autonomously (autonomy). They can also initiate some behaviors according to their surroundings (pro-activeness). We call those entities multiple objects. The conventional object-oriented models can not represent the multiple objects appropriately. In this paper, we propose a novel agent model, called EAMMO, to represent the multiple objects, and implement an interpreter for the description language of EAMMO in Java. Our agent model consists of three types of agents as follows: (1) an upper-agent describes an autonomous and dynamic object; (2) a lower-agent describes a reflective and static object; and (3) an environmental-agent describes the autonomous surroundings of the other agents. As an example, we model soccer players (a kind of multiple objects in real world) in EAMMO. Three kinds of soccer agents are constructed with different powers. We simulate the soccer games to confirm the effectiveness of our model.

キーワード：オブジェクトの多面性，エージェント，サッカー

1. はじめに

オブジェクト指向モデル⁽¹⁾はさまざまな分野で有効である。しかし、人間のように状況に合わせて役割を変えて振る舞う性質(多面性)をもつ実体をモデル化することは困難である^{(2)~(5)}。これはオブジェクトの多面性⁽⁶⁾の問題として知られている。本論文では、次のような性質を持つ実体に注目する。

多面性： 状況に合わせて異なる役割を持つ性質。例えば、サッカー選手が「ドリブルをする」、「パスをする」、「シュートをする」という役割を持つこと。

自律性： 時間や場所、状況に合わせて自分自身で役割や振る舞いを変化させる性質。例えば、サッカー選手がボールを持っていて、敵ゴールの前にいるなら自分のポジションがどこであろうと、「シュートをする」役割を自分で決定できること。

自発性： 環境の状態(例えば、経過時間)が変化する中で、オブジェクトが自分自身で行動を起こす性質。例えば、サッカー選手が残り時間によって自分自身で戦略を変更すること。

この三つの性質を持つ実体をモデル化したオブジェクトを多面的オブジェクトと呼ぶことにする。

これまでもオブジェクトの多面性の問題を解決するために Views⁽²⁾, Aspects⁽³⁾, Roles⁽⁴⁾, Deputy⁽⁵⁾ のようなモデルが提案されてきた。これらのモデルは多面性を表現することはできるが、自律性、自発性を持たないため、多面的オブジェクトを表現するには不十分である。また、プログラミング言語 JAVA⁽⁷⁾ でもオブジェクトの多面性のために「インタフェース」を実装している。しかし、多面性を表現できるのは、プログラム実行時の最初の一回だけである。多面的オブジェクトは実時間上で自律的、自発的に行動を決定する必要があるため、これも不十分である。そこで我々は次の三つのタイプのエージェントから構成されるエージェントモデル EAMMO(an Environmental Agent Model for Multiple Objects)を提案する。

下級エージェント： 多面性を持たず反射的な行動をするエージェント。サッカーゲームをモデル化するとき、サッカーのボールなどをモデル化するのに適している。

上級エージェント： 多面性を持ち、自律的、自発的に行動

動できるエージェント。同様の例で考えると、サッカー選手をモデル化するのに適している。

環境エージェント： 多面性を持ち、自律的、自発的な行動のできるエージェント。環境を表現するためのエージェントであるため、その環境に含まれるエージェントの情報（例えば、プレイヤーの位置など）を持つ。同様の例で考えれば、サッカーフィールドをモデル化するのに適している。

これらのエージェント全てのメッセージパッシングによる相互作用を通して、多面的オブジェクトは表現可能となる。本論文ではさらにこのモデルの記述処理系のプロトタイプを示し、その処理系による実験の結果を示す。

本論文は、第2章でエージェントモデルの説明に要する基本概念について述べ、続く第3章でエージェントモデル EAMMO とその記述処理系について述べる。第4章では処理系による実験としてサッカーエージェントの例について述べ、第5章で本論文のまとめを述べる。

2. 基本概念

本論文では、従来のオブジェクトをレギュラーオブジェクトと呼ぶことにする。レギュラーオブジェクトは識別子、属性のリスト、メソッドの集合によって表現される。属性はレギュラーオブジェクトの内部状態を表現するものであり、次のように定義される。

〔定義1〕（属性） 属性名 $\langle \text{attribute_name} \rangle$ と属性値 $\langle \text{attribute_value} \rangle$ の対から、一つの属性は構成される。

```
 $\langle \text{attribute} \rangle$   

 $::= \langle \text{attribute\_name} \rangle (\langle \text{attribute\_value} \rangle)$ 
```

また一つのオブジェクトの持つ属性のリストは次のように定義できる。

```
 $\text{Attributes}\{\langle \text{attribute} \rangle, \dots, \langle \text{attribute} \rangle\}$ 
```

ここで、 Attributes はそれに続くものが属性のリストであることを示すキーワードである。

なお、状態とは一つ以上の属性の集合によって表現されるものである。

識別子は各オブジェクトを識別するために用いる。また、識別子はオブジェクトごとに異なり唯一のものである。これは次のように定義できる。

〔定義2〕（識別子） 各オブジェクトは必ず唯一の識別子を持つ。

```
 $\langle \text{identifier} \rangle ::= \text{Object}(\langle \text{object\_name} \rangle)$ 
```

ここで Object はそれに続くものが識別子であることを示すキーワードである。

また、オブジェクトはメソッドの集合を持つ。各メソッドはオブジェクトの持つ手続きや関数を表す。これを定義すると次のようになる。

〔定義3〕（メソッド） メソッド名 $\langle \text{method_name} \rangle$ とメソッドの内容 $\langle \text{method_body} \rangle$ の対から、一つのメソッドは構成される。

```
 $\langle \text{method} \rangle ::= \langle \text{method\_name} \rangle (\langle \text{method\_body} \rangle)$ 
```

ここで、 $\langle \text{method_body} \rangle$ は適当な言語で記述された手続きや関数である。オブジェクトの持つメソッドの集合は次のように定義する。

```
 $\text{Methods}\{\langle \text{method} \rangle, \dots, \langle \text{method} \rangle\}$ 
```

ここで Methods は次に続くものがメソッドの集合であることを示すキーワードである。

これまでの定義を用いてレギュラーオブジェクトは次のように記述できる。

〔定義4〕（レギュラーオブジェクト） レギュラーオブジェクトは、識別子、属性のリスト、メソッドの集合から構成される。したがって次のように記述できる。

```
 $\langle \text{regular\_object} \rangle$   

 $::= \text{Object}(\langle \text{object\_name} \rangle),$   

 $\text{Attributes}\{\langle \text{attribute} \rangle, \dots, \langle \text{attribute} \rangle\},$   

 $\text{Methods}\{\langle \text{method} \rangle, \dots, \langle \text{method} \rangle\}]$ 
```

このオブジェクトは、自分の役割を決定するための機構やルールを持たないため、多面性を自分で決定することはできない。

オブジェクト指向において、オブジェクト間のメッセージは重要な概念である。エージェントもまたメッセージを介してコミュニケーションをおこなう。メッセージは次のように定義できる。

〔定義5〕（メッセージ） メッセージは、メッセージの内容、送り手、受け手の三つの要素で構成される。

```
 $\langle \text{message} \rangle$   

 $::= \text{Message}(\langle \text{a\_content\_of\_the\_message} \rangle,$   

 $\langle \text{sender} \rangle, \langle \text{receiver} \rangle)$ 
```

ここで Message はそれに続くものがメッセージに含まれる要素であることを示すキーワードである。また、 $\langle \text{a_content_of_message} \rangle$ はメソッド名、またはメソッドを起動するためのキーワードが記述される。

ここで、レギュラーオブジェクトを拡張することによって多面的オブジェクトを定義する。これまでの述べたことから、多面的オブジェクトを簡単に表現すると次のようになる。

多面的オブジェクト

= レギュラーオブジェクト + 自律性 + 自発性

つまり多面的オブジェクトは自律的で自発的なレギュラーオブジェクトである。したがって、多面的オブジェクトを

表現するためには、レギュラーオブジェクトに自律性と自発性を与えなければならない。

この性質を与えるために、状態決定プロセッサと動作決定プロセッサを定義する。このプロセッサは各多面的オブジェクトに含まれ、次の二段階のステップで、自分自身で自律的に振る舞いを決定できる。

(1) 状態決定プロセッサによって、自分の内部状態を更新する。

(2) その内部状態に基づき動作決定プロセッサによって、適切なメソッドを決定する。

更新された内部状態は、その多面的オブジェクトの役割(一つの性質)を表現し、その役割に基づいて振る舞いを決定する。

〔定義6〕(状態決定プロセッサ) 状態決定プロセッサは、計算機上の一つのプロセスであり、そのオブジェクトの属性のリストやメソッドを他のオブジェクトとの関係、環境の現在の状態、自分の現在の状態から、自分の新しい状態へ内部状態を更新する。

```

<status_processor>
::= Status{<relations>,<roles>}

<relations> ::= (<relation>,...,<relation>)
<relation> ::= Relation(<relation_name>)
               ←Object(<object_name>)

<roles> ::= (<role>,...,<role>)
<role> ::= <attribute_name>(<attribute_value>)
           ←Relation(<relation_name>),
           Own(<attributes>),
           Env(<attributes>))

Own(<attributes>) ::= Own{<attribute>,
                        ...,<attribute>}

Env(<attributes>) ::= Env{<attribute>,
                        ...,<attribute>}

```

ここで、Status, Relation, Own, Env はキーワードである。Status は次に続くものが状態決定のためのルールであることを示し、Relation は関係を制限するためのルールであることを示す。また、Own は続く属性のリストが現在の自分の属性であることを示し、Env も同様に、それが現在の環境の状態であることを示す。

<relation> の右辺の ← は、← の右辺のオブジェクトがメッセージの送り主オブジェクトであったとき <relation_name> で示される関係が成立することを示す。<role> でも同様に ← の右辺の条件が満たされるとき、左辺の <attribute_name> の値が <attribute_value> に更新されることを示す。

〔例1〕 オブジェクト Ito の状態決定プロセッサは次のように記述できる。

```

Status { (Relation(offense) ← Object(Nakagawa)
         Relation(offense) ← Object(Du),
         Relation(defense) ← Object(Kawai),
         Relation(defense) ← Object(Matsui),
         : : ),
         (role(passer)
         ←Relation(offense),
         Own(ball_keeping(true)),
         Env(forward_sight(敵側選手はいない)),
         : : ),
        }

```

これは次のような意味である。もし(チームのメンバーである) Nakagawa と Du を相手オブジェクトであると Ito が判断すれば、Ito と相手オブジェクト(Nakagawa, Du) の間に関係 offense が成立する。同様に、相手オブジェクトが(敵チームの) Kawai か Matsui であれば、関係 defense が成立する。また、Ito が相手オブジェクトとの間に関係 offense を持ち、ボールを自分が持っていて、かつ自分の前方(forward_sight)に敵チームの選手が誰もいないなら、自分の内部状態である属性名 role の属性値を passer に更新する。

〔定義7〕(動作決定プロセッサ) 動作決定プロセッサは計算機上の一つのプロセスであり、状態決定プロセッサが更新した自分の状態、現在の環境の状態、このオブジェクトが受け取ったメッセージによって適切なメソッドを選択し起動する。

```

<action_processor> ::= Actions{<action>,
                                ...,<action>}

<action>
::= (<method_name>
     ← Own(<attributes>),
     Env(<attributes>),
     Msg(<a_content_of_the_message>))

```

ここで Actions, Msg はキーワードである。Msg はその後が続くのが定義5で定義したメッセージの内容であることを示すものである。また ← の右辺の条件を満足すると、左辺のメソッド <method_name> が起動される。

〔例2〕 これはあるオブジェクト Ito の動作決定プロセッサの記述例である。

```

Actions{pass ← Own(role(passer)),
        Env(distance_to_the_goal(50)),
        Msg(pass),
        :      :      :
}

```

これは Ito が属性名 `role` に対して属性値 `passer` を持ち、ゴールの近くにいて、かつメッセージに `pass` というキーワード（またはメソッド名）が含まれていればゴールへボールをシュートするという意味である。

3. EAMMO

EAMMO(an Environmental Agent Model for Multiple Objects) は多面的オブジェクトの表現を目的としたマルチエージェントモデルである。

多面的オブジェクトを表現するためにはレギュラーオブジェクトに対して、自律性と自発性を与える必要がある。このため、本モデルでは状態決定プロセッサと動作決定プロセッサを導入することによって自律性と自発性を与える。この二つのプロセッサの動作には先に述べた通り環境の状態が必要であるため、エージェントモデルには環境が必要となる。環境とはエージェントをとりまく空間を指す。その中には時間のように自発的な要素や温度の変化のような自律的な要素も含まれる。したがって環境もエージェントであると考えerる必要がある。

また実世界では、人間のように複雑に振る舞うものもあれば道具のように単純な振る舞いしかできないものもある。したがって、これらをモデル化するために、それぞれに適したモデリング能力を持つエージェントを用意する必要がある。例えばハサミをモデル化するような単純なエージェントは自律的で自発的な振る舞いをする必要はない。逆に人のように複雑なものをモデル化するエージェントには自律的で自発的な振る舞いが要求される。したがって我々のエージェントモデルでは反射的な振る舞いをする下級エージェント、自律的で自発的な振る舞いをする上級エージェント、他のすべてのエージェントに影響を与える（例えばブロードキャストメッセージを送れるような）環境エージェントの三つのタイプのエージェントによって構成される。この三つのエージェントの関係を示したのが図 1 である。上級エージェントや下級エージェントは環境エージェントに含まれる形で存在する。これら三つのタイプのエージェントの相互作用によって多面的オブジェクトを表現できる。

〈3・1〉 下級エージェント

〔定義 8〕（下級エージェント） 下級エージェントは識別子、属性のリスト、メソッドの集合、動作決定プロセッサから構成される。

```

(lower-agent)
::= Agent((agent_name)),
    [Attributes{<attribute>, ..., <attribute>},

```

図 1 三つのタイプのエージェント間の関係
Fig. 1. Relation among three types of agents

図 2 下級エージェント
Fig. 2. A Lower-Agent

```

[Methods{<method>, ..., <method>},
 [Actions{<action>, ..., <action>}]]]

```

その概念図を図 2 に示す。下級エージェントはレギュラーオブジェクトに動作決定プロセッサを加えたものである。他のエージェントからのメッセージに対して反射的に動作することができる。

〈3・2〉 上級エージェント

〔定義 9〕（上級エージェント） 上級エージェントは多面性を持つ、自律的で自発的なオブジェクトとして定義できる。上級エージェントは識別子、属性のリスト、メソッドの集合、状態決定プロセッサ、動作決定プロセッサによって構成される。

```

(upper-agent)
::= Agent((agent_name)),
    [Attributes{<attribute>, ..., <attribute>},
    [Methods{<method>, ..., <method>}
    [Sequential_methods{<sequential_method>,
        ..., <sequential_method>}]],
    [Status{<relations>, <roles>}],
    [Actions{<action>, ..., <action>}]]

```

ここで `Sequential_methods` はメソッド列を示す予約語であり、`<sequential_method>` は連続したメソッドを示す。これは次のように定義される。

```

(sequential_method)
::= (<sequential_method_name>(<method_name>,
    ..., <method_name>))

```

図3 上級エージェント
Fig. 3. An Upper-Agent

```

← Own((attributes)),
  Env((attributes)),
  Msg((a_content_of_the_message)))

```

メソッド列 (sequential_method_name) が決定されると、これに続く (method_name) が順番に実行される。上級エージェントの概念図を図3に示す。上級エージェントは下級エージェントに状態決定プロセッサを加えたものである。状態決定プロセッサは属性やメソッド列を操作でき、他のエージェントとの関係、現在の環境の状態、現在の自分の内部状態から内部状態を適切なものに更新できる。動作決定プロセッサは状態決定プロセッサの結果と、環境の状態、他のエージェントからのメッセージを反映して実際に適切なメソッドを起動する。

〈3・3〉環境エージェント

〔定義10〕(環境エージェント) 環境エージェントは識別子、属性のリスト、メソッドの集合、状態決定プロセッサ、動作決定プロセッサ、環境データベースから構成される。

```

(environmental-agent)
::= Agent((agent_name)),
  [Attributes{(attribute),..., (attribute)},
  [Methods{(method),..., (method)},
  [Sequential_methods{(sequential_method),
    ..., (sequential_method)}]]],
  [Status{(relations), (roles)}],
  [Actions{(action),..., (action)}],
  [Env_db{(data),..., (data)}]

```

ここで、Env_db は続く (data) のリストが環境データベースのデータであることを示すキーワードである。これは次のように定義される。

```

(environmental_database)
::= Env_db{(data),..., (data)}

```

図4 環境エージェント
Fig. 4. An Environmental-Agent

```

(data)
::= (Agent((agent_name)),
  Attributes{(attribute),..., (attribute)})

```

その概念図を図4に示す。このエージェントは上級エージェントに環境データベースを加えたものである。上級エージェント、下級エージェントは視覚的センサーを持たない。しかし、環境エージェントが環境データベースにより、視覚情報を提供する視覚補助的な役割を果たす。また、上級エージェント、下級エージェントが自律的、自発的に動作するための要素を属性として持つことができる。ただし、これらの属性は扱いが異なる。例えば、時間に依存した属性に関しては環境エージェントによって操作が可能であるが、実際の時間そのものについては操作はできない。具体的には、サッカーの試合において実際の時間に依存して残り時間を更新していけるが、実際の時間そのものは変更できない。時間以外の特別な要素（例えばOSの状態など）についても同様である。また、環境データベースにより、環境内へのブロードキャストやマルチキャストをおこなうことができる。エージェントが別の環境エージェントに移動するときは、この環境データベースに対して登録や削除がおこなわれる。

一つの環境エージェントは、その環境エージェント内に含まれる他のエージェントの情報を管理する。しかし、その環境エージェントに含まれる他の環境エージェントに含まれるエージェントを管理するわけではない。したがって、EAMMO は集中と分散の両方の性質を併せ持つエージェントモデルであり、連邦型マルチエージェントシステム⁽¹⁰⁾に分類できる。

〈3・4〉記述例 ここで、Ito という名前のサッカープレイヤーの例を示すことにする。このプレイヤーの持つ戦略を表1に示した。表中の「-」は「場所」、「ボール」などの条件には束縛されない、つまり無視して構わないことを表す。サッカープレイヤーはゲームの状況や自らのポジション、サッカーフィールド上での位置に合わせてさまざまな役割を持つため、多面的オブジェクトの良い例である。

このプレイヤーを上級エージェントとしてモデル化すると、図5のように記述できる。

表 1 サッカープレイヤーの戦略
Table 1. A strategy of a soccer player

関係	オフェンス				ディフェンス
場所	ゴールエリア内		ゴールエリア外		—
ボール	○	×	○	×	—
戦略	シュート	ゴール前へ	パス	ゴール前へ	ボールを追う
例外処理	—	—	ゴールが近い → シュート	—	—

```

Agent(Ito),
Attributes(team(A), number(10),role(track_ball), ball(true)),
Methods(shoot(...),pass(...),goto_goal(...),track_ball(...)),
Status((Relation(offense) -> offense,
        Relation(defense) -> defense),
        (role(shoot) -> Relation(offense),
          Env(where(goal_area)),
          Own(ball(true))),
        role(pass) -> Relation(offense),
          Env(where(not_goal_area)),
          Own(ball(true))),
        role(goto_goal) -> Relation(offense),
          Env(),Own(ball(false))),
        role(track_ball) -> Relation(defense),
          Env(),Own()),
Actions(shoot -> Own(role(shooter)),Env(),Msg(doing),
        shoot -> Own(role(passer)),Env(dist(short)),Msg(doing),
        pass -> Own(role(passer)),Env(),Msg(doing),
        goto_goal -> Own(role(goto_goal)),Env(),Msg(doing),
        track_ball -> Own(role(track_ball)),Env(),Msg(doing))

```

図 5 サッカープレイヤーの記述

Fig. 5. Description of a soccer player

このプレイヤーがボールを持っていて、そのチームがオフェンスであると仮定する。さらに、このプレイヤーがゴールエリア外にいて (`where(not_goal_area)`)、実際に行動を起こすときには、自分と敵チームのゴールとの距離が短くなるとする。このとき、プレイヤー Ito は自分の内部状態を `role(pass)` に更新する。そして、この内部状態と `Dist:short` という環境の状態に基づいて `shoot` という振る舞い (メソッド) を決定する。ここでこのプレイヤーがパスをする役割を担いながらシュートできる点に注意する。上級エージェントは状態の決定と動作の決定の二段階で振る舞いを決定する。このため、パスをするという振る舞いからシュートをするという振る舞いに変更できる。つまり、我々のモデルはこのような例外処理も表現できる。

〈3・5〉記述処理系 我々はサッカーエージェントの記述とシミュレーションを効率良く行なうために記述処理系を JAVA 言語を用いて実装した。

各エージェントは基本的に次のような三つのユニットから構成される。

- (1) コミュニケーションユニット：このユニットは他のエージェントとコミュニケーションし、エージェントに理解できる形式にメッセージを変換する。
- (2) 仕様ユニット：このユニットはエージェントの自律的、自発的な振る舞いに必要とされるルールの集

図 6 エージェントの基本構造

Fig. 6. Common structure of agents

合である。

- (3) 操作ユニット：このユニットはインタプリタとして実装され、仕様ユニットの内容を修正することや、仕様ユニットの記述内容から適切なメソッドを決定し実行することができる。

図 6 は、これら三つのユニットがどのように EAMMO のエージェントを構成するかを示している。この構造は各 EAMMO エージェントの基本構造である。

また各エージェントは次に示すクラス階層を構成する。

- (1) **Agent Class**：エージェント識別子や属性のリストなど、全てのエージェントに共通する性質を持つクラス。
- (2) **Lower-Agent Class**：Agent Class を継承するクラスで、動作決定プロセッサに関する部分を追加したクラス。
- (3) **Upper-Agent Class**：Lower-Agent Class を継承するクラスで、状態決定プロセッサに関する部分を追加したクラス。
- (4) **Environmental-Agent Class**：Upper-Agent を継承するクラスで、環境データベースに関する部分や、環境内にブロードキャストするメソッドに関する部分を追加したクラス。

この階層は Agent Class を根とした継承による階層である。各エージェントはこれらのクラスのインスタンスとして生成されることになる。

4. サッカーエージェント

EAMMO の記述処理系を用いて RoboCup (Robot World Cup) のサッカーエージェントを記述する。

RoboCup とは人工知能を持った複数のロボットによるサッカーの大会であり、チェスに代わる人工知能のランドマーク問題として注目されている。サッカーは 11 人対 11 人のスポーツであるので、全部で 22 個のクライアントプログラムと、それを統括するサッカーサーバから構成される。したがって試合をするときには、各チーム 11 個のプログラムを並列に動作させ、個々のプログラム (プレイヤー) はサッカーサーバから得られる情報から自分の動作を判断する。また、プレイヤーの視界は制限 (前方 60 度) されており、距離によって情報が欠落していくなど、不完全な情報しか与えられない。したがって、自分の周囲の状況を的確に判断し、振る舞いを決定する必要がある。また、お互

いに協調してチームプレイしていく必要がある。ここではこれ以上詳しく述べないが、RoboCupの詳しい情報は参考文献⁽⁸⁾⁽⁹⁾に述べられている。

本モデルによるモデル化に際し、サッカープレイヤーの環境にあたるサッカーフィールドをサッカーサーバとしてRoboCup側から与えられているため、それに応じたインタフェースを用意する必要がある。このためEAMMOのオブジェクトクラスのサブクラスを設計する。

- (1) **SoccerEAgent Class**: Environmental-Agent Classを継承するクラス。サッカーサーバのインタフェースとしてこのクラスを利用するため、サッカーサーバへの入出力メソッドを持つ。
- (2) **SoccerUAgent Class**: Upper-Agent Classを継承するクラス。このクラスはSoccerEAgent Classとのコミュニケーションメソッドを持つ。
- (3) **SoccerLAgent Class**: Lower-Agent Classを継承するクラス。このクラスもSoccerEAgent Classとのコミュニケーションメソッドを持つ。

サッカーサーバとのインタフェースである環境エージェント(**SoccerEAgent Class**)がサッカーエージェントにとっての環境となる。この**SoccerEAgent Class**(のインスタンス)は各エージェントが持つ。この環境は、一定時間で更新されていく必要があり、視覚情報も聴覚情報も自発的に更新する必要があるため、自律的で自発的なエージェントという形態が適している。

現在の実装では上級エージェントと下級エージェントは直接、コミュニケーションできない。これらのエージェントは環境エージェントを通してコミュニケーションすることになる。これらのクラスに基づいて三種類のサッカープレイヤーを構築した。

- (1) **下級プレイヤー**: 下級エージェントと環境エージェントから構成されるプレイヤー。下級エージェントを用いて反射的にプレイする。
- (2) **上級プレイヤー**: 上級エージェントと環境エージェントから構成されるプレイヤー。状態決定プロセッサによって決定される内部状態に基づいてプレイする。
- (3) **複合プレイヤー**: 下級エージェント、上級エージェント、環境エージェントから構成されるプレイヤー。このプレイヤーは上級エージェントによる自律的で自発的な性質と下級エージェントによる反射的な性質を合わせ持つ。

これらの三種類のプレイヤーを使って三つのチーム(下級チーム、上級チーム、複合チーム)を構成する。それぞれのエージェントの振る舞いを確認する意味で、これらのチームを用いて試合を行なった。試合の様子を示したのが図7である。

表2は複合チームと上級チームの対戦結果である。ここで、複合チームに含まれる上級エージェントと上級チームに含まれる上級エージェントは、同一のエージェントを用いた。この表からもわかる通り、複合チームが72%の確率

図7 サッカーの試合の様子
Fig. 7. A Soccer Game

で勝利した。複合チームでは、上級エージェントが100ms内でプレイを終了しない場合、下級エージェントが反射的なプレイを行なう。また、複合エージェントでの上級エージェントと下級エージェントの処理の比率は5:1であった。一方、上級チームは上級エージェントのみで自律的、自発的にプレイをする。この場合、上級エージェントのプレイに時間がかかってしまう場合には、得点のチャンスなどを逃してしまう結果となる。したがって、複合チームの勝因は上級チームより高速かつ正確に振る舞うことができたためである。

表3は複合チームと下級チームの対戦結果である。複合チームはすべての試合で負けなかった。下級チームでは、すべてのプレイヤーがボールを追い、ボールの周囲に集まった。一方、複合チームはサッカーフィールド全体に広がって自律的にプレイをした。したがって複合プレイヤーの方がパス中心の組織的な試合をできたことが勝因である。

ここでは上級チームと下級チームの対戦結果を示さないが、複合チームと下級チームの対戦と同様に、上級チームが圧勝する結果が得られた。

これらの結果から、複合チームがわれわれの実験では最も強いという結果を得ることができた。したがってRoboCupにおいては、一定のインターバルで適切に動作することが非常に重要であるといえる。

われわれの記述処理系にはもう一つ重要な利点がある。それは実際に試合をしながら、動的に上級エージェント、下級エージェントの戦略を改良することができる点である。個々のエージェントを記述してチームプレイを成立させるため、実際に動かしながらチームプレイを調整できることは非常に重要である。我々の処理系は、環境エージェント

表2 複合チーム (HPT) と上級チーム (UPT) の対戦結果

Table 2. Hybrid-Player Team(HPT) vs. Upper-Player Team(UPT)

複合チームのキックオフ					
HPT	UPT	結果	HPT	UPT	結果
3	1	HPT	4	3	HPT
3	1	HPT	3	1	HPT
2	0	HPT	6	3	HPT
4	1	HPT	5	4	HPT
3	4	UPT	4	5	UPT
3	1	HPT	4	4	DRAW
3	4	UPT	3	1	HPT
4	3	HPT	5	6	UPT
2	2	DRAW	8	1	HPT
5	1	HPT	4	6	UPT
5	1	HPT	3	2	HPT
2	3	UPT	4	1	HPT
6	1	HPT			
上級チームのキックオフ					
HPT	UPT	結果	HPT	UPT	結果
4	1	HPT	4	0	HPT
4	1	HPT	7	1	HPT
4	1	HPT	3	4	UPT
2	2	DRAW	5	0	HPT
8	1	HPT	4	2	HPT
4	1	HPT	3	2	HPT
5	1	HPT	4	2	HPT
3	3	DRAW	2	2	DRAW
6	4	HPT	2	1	HPT
4	2	HPT	6	2	HPT
5	3	HPT	4	3	HPT
2	3	UPT	3	3	DRAW
5	2	HPT			

複合チームの勝率 : 72%
上級チームの勝率 : 16%

表3 複合チーム (HPT) と下級チーム (LPT) の対戦結果

Table 3. Hybrid-Player Team(HPT) vs. Lower-Player Team(LTP)

複合プレイヤーのキックオフ					
HPT	LPT	結果	HPT	LPT	結果
6	3	HPT	5	1	HPT
6	3	HPT	5	2	HPT
7	3	HPT	5	2	HPT
6	3	HPT	5	3	HPT
7	1	HPT	5	2	HPT
5	3	HPT	7	2	HPT
5	2	HPT	4	4	DRAW
6	3	HPT	5	1	HPT
6	3	HPT	7	2	HPT
5	2	HPT	6	2	HPT
7	2	HPT	5	2	HPT
4	2	HPT	5	2	HPT
6	3	HPT			
下級プレイヤーのキックオフ					
HPT	LPT	結果	HPT	LPT	結果
5	2	HPT	6	1	HPT
5	3	HPT	4	2	HPT
5	1	HPT	5	2	HPT
4	4	DRAW	4	2	HPT
5	1	HPT	5	3	HPT
6	2	HPT	4	3	HPT
5	3	HPT	4	1	HPT
5	3	HPT	5	2	HPT
4	1	HPT	5	3	HPT
6	3	HPT	4	2	HPT
4	4	DRAW	7	3	HPT
6	2	HPT	5	1	HPT
4	2	HPT			

複合チームの勝率 : 94%
下級チームの勝率 : 0%

がサッカーサーバと上級エージェント、下級エージェントの間のインタフェースとして存在する。サッカープレイヤーにとって環境として存在する環境エージェントはサッカーサーバにとっては直接通信しているエージェントであるため、サッカープレイヤーとして認識される。したがって、環境エージェントのプロセスを停止しなければサッカープレイヤーは動作中であるとサッカーサーバに認識される。このためサッカープレイヤーの動きを確認しながら上級エージェント、下級エージェントの記述を改良して推論プロセスのみを修正することが可能である。つまり、我々の処理系は繰り返しチームプレイや戦略を改良するような場合には非常に有効であるといえる。

5. ま と め

本論文では、オブジェクトの多面性を表現するためのエージェントモデル EAMMO と、その処理系について述べた。我々のモデルやシステムが RoboCup のサッカーエージェントに適していることを実験を通して確認できた。また、我々の処理系はサッカーエージェントの振る舞いをみながら、修正できるので、サッカーエージェントの実験環境と

しては非常に有効であることも確認できた。

我々は処理系を JAVA 言語を用いてインタプリタシステムとして実装した。オブジェクト指向言語としての JAVA は本モデルの実装に適している。しかし、その実行速度はまだ遅く、またインタプリタシステムを採用したことも実行速度を落とす原因となった。現在、仕様ユニットの記述を特定の記述に変換する処理系を別に用意することや、スレッドを導入することによってシステムの動作速度の向上をおこなっている。また別言語への移植も検討中である。本論文では、単純なルールベースの推論機構を用いている。現在、学習を用いたシステムや高度な書換えシステムと、その保護機構の導入も検討中である。

(平成 10 年 4 月 23 日受付, 同 10 年 9 月 9 日再受付)

文 献

(1) J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorenson: *Object-Oriented Modeling and Design*, Prentice Hall(1991).
(2) J. J. Shilling: "Three Steps to Views: Extending the Object-Oriented Paradigm", *OOPSLA '89 Proceedings*, pp.353-361(1989).

- (3) J. Richardson, P. Schwarz: "Aspects: Extending Objects to Support Multiple, Independent Roles", *ACM SIGMOD '91*, pp.298-307(1991).
- (4) A. Albano, R. Bergamini, G. Ghelli, R. Orsini: "An Object Data Model with Roles" *Proceedings of the 19th VLDB Conference*, pp.39-51(1993).
- (5) Y. Kambayashi, Z. Peng: "Object Deputy Model and Its Applications" *Proceedings of Fourth International Conference on Database System for Advanced Applications(DASFAA '95)*, pp.1-15(1995).
- (6) S. Tsukada, T. Sugimura: "MAC-model: an extended object-oriented data model for multiple classification," *Computer Software, Journal of JSSST*, 11(5), pp.400-413(1994).
- (7) G. Cornell, C. S. Horstmann: コア JAVA, アスキー出版局(1997).
- (8) I. Noda: "Soccer Server Manual Rev. 1.03" <http://ci.etl.go.jp/noda/soccer/manual.newest/main.html>
- (9) H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara: "RoboCup A Challenge Problem for AI", *AI MAGAZINE*, SPRING 1997, pp.73-85(1997).
- (10) S. Bird: "Toward a Taxonomy of Multi-agent System" *int. J.M-M Stud.*, Vol.39, pp.689-704(1993).
- (11) 伊藤暢浩, 中川浩一, 石井直宏:「多面的性質を表現可能な環境のエージェントモデル EAMMO」, 平成9年度電気関係学会東海支部連合大会論文集, p.310(1997).
- (12) 中川浩一, 伊藤暢浩, 石井直宏:「多面的なエージェントを用いたサッカーエージェントの構築」, 平成9年度電気関係学会東海支部連合大会論文集, p.310(1997).

石井直宏 (正員) 1940年3月23日生。1968年東北大学

IEEE 各会員。

工学部電気工学科卒業。1973年同大学院工学研究科博士課程修了。工学博士。東北大学医学部助手, 名古屋工業大学電気情報工学科助教授を経て, 現在, 名古屋工業大学知能情報システム学科教授。この間, しきい値論理, 生体情報処理, 非線形システム解析, ニューラルネットワークの研究に従事。電気学会, 電子情報通信学会, 神経回路学会,

伊藤暢浩 (非会員) 1970年9月3日生。1994年3月名

古屋工業大学電気情報工学科卒業。1996年3月同大学大学院博士前期課程修了。同年4月同大学院博士後期課程入学。現在, 3年次在学中。オブジェクト指向技術, エージェント指向技術に興味を持つ。電子情報通信学会, 情報処理学会, 人工知能学会各会員。

中川浩一 (非会員) 1976年1月24日生。1998年3月名

古屋工業大学知能情報システム学科卒業。同年4月同大学大学院工学研究科博士前期課程電気情報工学専攻入学。現在, 1年次在学中。エージェント指向技術に興味を持つ。人工知能学会会員。

杜小勇 (非会員) 1963年8月22日生。1983年中国杭

州大学理学部数学科卒業。1988年中国人民大学大学院情報学専攻修士課程修了。1993年より名古屋工業大学に留学。1997年名古屋工業大学大学院工学研究科博士後期課程電気情報工学専攻修了。工学博士。同年4月同大学知能情報システム学科助手。現在に至る。データベースおよび人工知能の研究に従事。情報処理学会, 電子情報通信

学会各会員。