

# A Role-Based Approach to Robot Agent Team Design

Haibin Zhu, *Senior Member, IEEE*

**Abstract**— Robot team work is a typical collaborative activity. How to organize robot teams and design robots are important and complex tasks.

This paper explores the properties of agents and agent systems, introduces and revises the role-based collaboration model E-CARGO, demonstrates the essential design concerns of roles, agents, agent collaboration and agent teams, and proposes that roles can be used to form a fundamental structure, i.e., role nets, to design robot agent teams. Finally, it points out the research topics of role-based design for robot agent systems.

## I. INTRODUCTION

AGENT collaboration is an exciting research topic even though there are a variety of arguments on agent definitions. To understand agent collaboration, we need to clarify what agents mean in this paper. Autonomous agents are considered as a system situated with a part of an environment that senses to accomplish its own agenda and effect what it senses in the future [4, 5]. In a collaborative agent system, agents should be [8]

- Active, i.e., an agent might act by its internal states. Note that an object in its conventional meaning can only respond to the messages sent to it even though we assert that everything is an object;
- Autonomous, i.e., an agent is not controlled directly by people; and
- Collaborative, i.e., agents need to collaborate with others to accomplish a complex task.

Every agent is responsible for accomplishing a certain task. An agent can be considered as a self-contained object of some class, which does not belong to the problem domain although it does involve itself with that domain or environment.

This paper aims to describe the situation and the problems that might be encountered when designing roles for agents, such as robots. In the following discussion, we aim at a robot team as the problem domain. Therefore, in this paper, robots agents, and robot agents are synonyms.

Agents can interact with other agents or environments in a cooperative or a competitive way [1]. The coordination mechanism in a robot team should provide flexibility and adaptability so as to make the robots more efficient [3]. To design an agent that is good at cooperation is a difficult task. Roles are one of the ways to support the separation of

concerns and are a possible way to “divide and conquer” the difficulties of designing cooperative agents. Roles can enable a separation between algorithmic and interaction-related issues in developing agent applications and permit the reuse of solutions and experiences [1].

There are many possible roles for robot agents to play. To play a role means that the robot works according to the regulatory processes confined by the role. In a real soccer team, every person has different abilities such as speed, durance, weight, height, experiences, education, and intelligences. A human soccer player can only play very limited roles during a period of time. Robots can be considered as agents with the same power and abilities, such as stamina and intelligence expressed by batteries, Central Process Unit (CPU), memory, and sensors. The differences among robots when they are built can be ignored. Therefore, a robot soccer team is different from a human soccer team. Hence, to build a robot team, we can use a totally different strategy than a human soccer team. For example, a real team cannot afford the strategy of “total football” because it requires too much energy and stamina of the soccer players to continue their efforts for a whole game. That is to say, a robot soccer player can play more roles in a period of time if the CPU, memory and sensors of the robot are powerful enough. At the extreme point, we can even install all the possible roles in a soccer team on each robot.

This paper is arranged as follows: Section II briefly introduces the E-CARGO model; Section III describes how roles can be designed, Section IV discusses the agent design based on roles; Section V discusses the cooperation design among agents based on roles; Section VI demonstrates how an agent team is designed with roles; Section VII concludes the paper and proposes the topics for future research.

## II. REVISED E-CARGO MODEL FOR A ROBOT TEAM

We can specify an agent’s role with two parts: the service interface including incoming messages, and the request interface including outgoing messages (Fig. 1) [10, 11], where, the human icon expresses an agent. To support role-based collaboration, we developed the E-CARGO model [10, 11] and revise it to encourage people to contribute in collaboration [9]. A robot team  $\Sigma$  can be described as a 9-tuple  $\Sigma ::= \langle C, O, \mathcal{A}, \mathcal{M}, \mathcal{R}, \mathcal{E}, \mathcal{G}, s_0 \rangle$ , where  $C$  is a set of classes,  $O$  is a set of objects,  $\mathcal{A}$  is a set of robot agents,  $\mathcal{M}$  is a set of messages,  $\mathcal{R}$  is a set of roles,  $\mathcal{E}$  is a set of environments,  $\mathcal{G}$  is a set of groups, and  $s_0$  is the initial state of a collaborative system. With E-CARGO, roles, role players and role playing are the three aspects of system design and development.

Manuscript received March 10, 2006. This work was supported in part by the IBM Eclipse Innovation Grant and NSERC of Canada (No. 262075-06).

Haibin Zhu is with the Department of Computer Science and Mathematics, Nipissing University, 100 College Dr., North Bay, ON, P1B 8L7, Canada (phone: +1-705-474-3450 ext. 4434; fax: +1-705-474-1947; email: [haibinz@nipissingu.ca](mailto:haibinz@nipissingu.ca)).

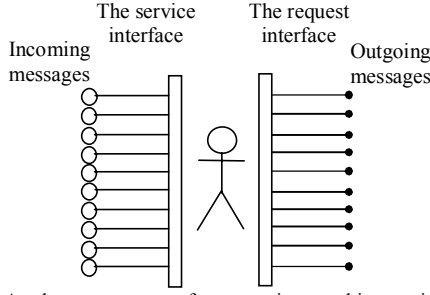


Fig. 1. A role as a wrapper of a person in a working environment.

An *object* is anything to be accessed by the robots in the team. A *class* is a template for a group of similar objects.

A message is defined as  $m ::= \langle n, v, l, \mathcal{P}, t, w \rangle$  where  $n$  is the identification of the message,  $v$  is null or the receiver of the message expressed by an identification of a role,  $l$  is the pattern of a message, specifying the types, sequence and number of parameters,  $\mathcal{P}$  is a set of objects taken as parameters with the message pattern  $l$ , where  $\mathcal{P} \subset O$ ,  $t$  is a tag that expresses any, some or all message,  $w$  is the weight for an agent to collect its credit for promotion, i.e., if an agent or its human user responds to this message the agent will get the weight and accumulate it to its credit.

A *role*  $r ::= \langle n, I, \mathcal{N}_a, \mathcal{N}_o, e_b, e_s, \mathcal{R}_w, \mathcal{R}_s, w \rangle$  where,  $n$  is the identification of the role,  $I ::= \langle \mathcal{M}_{in}, \mathcal{M}_{out} \rangle$  denotes a set of messages (wherein,  $\mathcal{M}_{in}$  expresses the methods or functions to respond the incoming messages,  $\mathcal{M}_{out}$  expresses a set of outgoing messages or message templates to roles),  $\mathcal{N}_a$  is a set of identifications of agents that are playing this role; and  $\mathcal{N}_o$  is a set of identifications of objects including classes, environments, roles, and groups that can be accessed by the agents playing this role,  $e_b$  and  $e_s$  are used to express the ability requirement,  $\mathcal{R}_w$  the super roles,  $\mathcal{R}_s$  the subordinate roles, and  $w$  the credit requirement.

An *environment* expresses a structure to build a group. It is specified by roles, the objects accessed by the roles and the cardinalities of agents playing roles.  $e ::= \langle n, \mathcal{B} \rangle$  where,  $n$  is the identification of the environment and  $\mathcal{B}$  is a set of tuples of role, number range and an object set,  $\mathcal{B} = \{ \langle n_r, q, \mathcal{N}_o \rangle \}$ . The number range  $q$  tells how many users may play this role in this environment and  $q$  is expressed by [lower, upper]. For example,  $q$  might be [1, 1], [2, 2], [1, 10], [3, 50], ... It states how many agents may play the same role  $r$  in the group. The object set  $\mathcal{N}_o$  expresses the complex objects accessed by the agents who play the relevant role. By “complex” we mean they are composed of other objects. The complex objects in  $\mathcal{N}_o$  are categorized as two kinds: singly owned or shared, i.e., one complex object in this set may be accessed by one agent or shared by many. In fact,  $\mathcal{N}_o$  expresses the resources for agents to access.

A *group* is a set of agents that are working on an environment, i.e., a set of agents assigned with roles in the relevant environment.

An *agent* is a special object that represents a robot in a

team. It is defined as  $a ::= \langle n, c_a, s, \mathcal{N}_r, \mathcal{N}_g, e_b, e_s, w, u, e \rangle$ , where,  $c_a$  is a special class that describes the common properties of users,  $n$  is the identification or name of the agent,  $s$  is the set of properties of the agent,  $\mathcal{N}_r$  means a set of identifications of roles the agent is playing, and  $\mathcal{N}_g$  means a set of identifications of groups the agent belongs to;  $e_b$  and  $e_s$  are used to express the ability,  $w$  the credit and  $u$  the workload.

With the above definitions, the agent is fully consistent with the idea that autonomous agents are situated in some environment emphasized by Franklin and Graesser [4] although they did not define what an environment is.

By E-CARGO, a robot team is built and activated by equipping robots with the following components:

- Objects are classified and instantiated for the team work.
- Roles are specified with the tasks and the cooperation requirement of the agent team.
- Agents are designed with essential mechanisms.
- Agent is attached with a group of roles to be played.
- An environment with associated roles and objects are specified;
- Agents (robots) form a group by playing roles in an environment of the group.
- Agents (robots) autonomously pursue the goals regulated by playing current roles, transferring active roles, pursuing future roles to cooperate with each other and reach their common goal [9].

### III. ROLE DESIGN

Stone and Veloso [7] believe that a role is composed of a specification of an agent internal and external behavior. They point out that assigning fixed positions to agents leads to several problems: short-term inflexibility in that the players cannot adapt their positions to the ball’s location on the field; long-term inflexibility in that the team cannot adapt to opponent strategy; local inefficiency in that players often get tired running across the field back to their positions after chasing the ball.

A role is considered as a specification of an agent’s internal and external behaviors [7]. A role can be considered as a function one or more agents perform during the execution of a cooperative task [2]. Agents are like people. If people want to play roles, they must have learnt how to behave, i.e., some pre-defined processes have been stored in the memory of the person. In this paper, we consider roles as special processes that accomplish a common task. That is to say, in robot agent teams, roles are not only the incoming and outgoing messages but also the implementations of them, especially the processing for the incoming messages.

Roles specify how a robot behaves at a specific context within a limited period. All the robots playing the same role use the same process to behave with the same specification. Robots behave differently in reality, because they are located at different places and have different environments and therefore are playing different roles.

With object-oriented principles, roles can be defined as classes and abide by the class inheritance and classification relationships as they are designed. Agents play role instances (Fig. 2), where, the rectangular bars express roles, the human icon expresses an agent, the solid line means *the current role* and dashed lines express *qualified roles*. By current role, we mean that the agent is currently playing the role. By qualified roles, we mean that the agent is qualified to play the roles but is not currently playing.

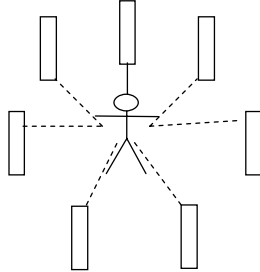


Fig. 2. An agent and its current and qualified roles.

If an agent has only one CPU, it can only play one current role at a time. If an agent is built on a computer with multiple CPUs, it may play many current roles at the same time. For a robot, if it is equipped with more than one CPU, it may play more than one current role but it may only play current roles without conflicts. For example, a robot can play the role of forward to attack the opponent team's gate with a ball, but it cannot at the same time play a defender role to block an opponent. The roles it can play at the same time are roles that produce physical actions such as moving, kicking, blocking and roles that do not produce physical actions such as measuring distances, computing the time of ball transfer, or computing the time used by an opponent.

There are also relationships among role instances. By differentiating role classes and role instances, we need to emphasize that the roles in an environment are role classes. Therefore, a group  $g$  is actually instantiating the role classes of an environment  $e$  and attaching the role instances to agents.

To design roles in an object-oriented programming language such as Java, we design roles in classes. These classes describe the services they can provide and assume rights to contact other roles. The rights assumed by this role are used to implement the process of each service.

A problem we may encounter in applying rights is how to identify the roles this role should ask for services. In traditional object-oriented programming languages, the outgoing messages are directed to the objects contained in the object of this class. In role-based collaboration, it is unreasonable to have a role instance contain other role instances even though this kind of inclusion is common for expressing object composition relationships.

Without role instance composition, it is required to build a visible scope for a role class. This scope includes all the visible role instances that this role class may have the right to ask for services. From the messages to role instances, agents will execute the method and behave according to the relevant

role instance. In a robot team, the scope of a role class can be simplified to be the role instances identified by port numbers that are related to a robot agent identified by an Internet Protocol (IP) addresses in a wireless network domain.

Therefore, based on the definition of role  $r ::= \langle n, I, \mathcal{N}_a, \mathcal{N}_o, e, \ell_s, \mathcal{R}_m, \mathcal{R}_f, w \rangle$  in E-CARGO, we could have a segment of a base class `C_Role` as follows:

```
abstract class C_Role
{
    //...
    Set Scope;
    //a set of role instances the role can access
    public abstract void instanceVariables();
    //return the state of the role
    //..
    public C_Role(){//a constructor of class C_Role
        //...
        Scope = new HashSet();
        //...
    };
    public void add_role(C_Role aRole){
        //...
        Scope.insert(aRole);
        //..
    }
    //...
}
```

From the class segment, we find that the scope is just an empty set when a role instance is created. To make a role instance really workable, it is required to add a role instance with the method `add_role(C_Role)` to the scope before the role instance can accept messages and process events.

#### IV. AGENT DESIGN

Agents are simulations of people or people's intelligent activities. Agent designers must have some representations of the knowledge that will be used by the agents and the system. In this sense, a robot agent is a machine that can provide many services. We need to emphasize that an agent should be equipped with at least a CPU, a memory and sensors. A cooperative agent should also be equipped with communication mechanisms such as a wireless networking card. In a robot soccer team, a robot is supposed to have basic abilities such as moving (forward or backward), turning (left or right), scanning (the ball, the cooperative agents and the opponent agents), kicking the ball (to the gate, to prevent the opponent's attacks), and controlling the ball. To play different roles is actually to apply different policies to activate these basic abilities. With these essential mechanisms, agents can play roles and cooperate with other agents by executing the methods specified in their class.

In agent systems, we need to emphasize that roles are behavior descriptions for agents. Agents behave according to their current role. An agents' potential is up to its qualified roles. With role-based approaches, we can help agents grow by designing more roles for them.

With pre-defined roles, we can simplify agent designs. We can have an agent attach role instances to let an agent develop. This attachment is actually to help agents learn.

With more and more roles attached, agents will be more and more intelligent.

In reality, people can change their status by playing a role. For example, they can obtain school credits by playing the role of student and collect working experiences in a company by playing the role of programmer. They should use a space in their memory or an external tool to store the credit they receive when playing the role of student, and another to store the experience obtained when playing the role of programmer. Using the same strategy, when a new role instance is attached to an agent, it should allocate a space to store new data relevant to the new role.

Therefore, designing agents is actually designing the class  $C_a$  of the model E-CARGO. The methods of class  $C_a$  is a set which covers all the incoming messages of the roles it plays.

In a robot team, a robot agent is allocated with a minimum required space when created. All the role classes are stored in the robot's memory. When the agent wants to play a new role, it instantiates the role class and attaches the role instance. The

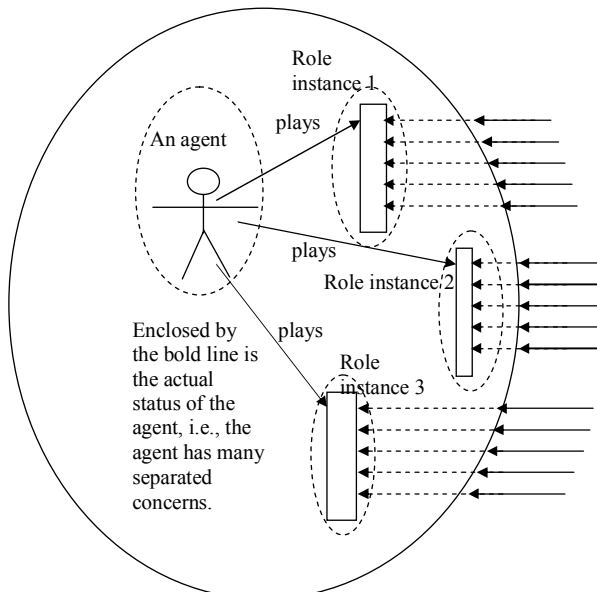


Fig. 3. The state of an agent is composed of both the agent instance and its role instances.

initial agent state and the new role instance construct the new state of the agent (Fig. 3).

As an autonomous agent, a robot must be able to decide what its favored current role is. We can call this decision making as role transfer. This policy should be combined with its own sensor states and the messages sent from other agents.

With a role instance, a message sent to a robot is transferred to a role instance and is finally processed by its role class. It is the role instance not the agent instance that is modified by the method relevant to a message (Fig. 3). In Fig. 3, messages are accepted by an agent (a robot) and then transferred to its relevant roles. To differentiate messages sent to an agent directly and those sent to its roles, we need to provide a way to have the agent transfer the messages to a relevant role. That is to say, an agent should be able to check its available roles and dispatch a message to a role instance. It

saves the effort of agent system designers to consider such message matching. They do not need to consider what roles an agent is playing.

The above method is reasonable because it is a fact that a message is sent and that a sensor event occurs. The robot agent should not pretend that these have not occurred. The active way is to deal with these events and accept the messages. What the agent can decide is how to deal with them. Here, roles play an important role for an agent to decide what to do.

The first decision is whether to transfer the current role. If the incoming messages and the sensor events are among the responsibilities of the current role, they are processed right away. If they are not among the responsibilities of the current role, we have two ways to process the message: one is to transfer the current role to a role that can process them; the other is to store the events and messages in a buffer and wait until the agent has done all the requests for the current role. To help make this decision, we need to set roles with priorities. If the incoming messages belong to a role with a higher priority, the current role should be transferred; otherwise, no transfer occurs.

As a summary, with pre-defined roles an agent design can be completed when it meets the basic requirement of processing, storing, communicating and other mechanical operations. Making the agent more intelligent is just assigning the agent predefined roles and setting the policy of role transferring, or also called the policy of dynamic role assignment [Chaim, Stone].

With the definition of agent in E-CARGO, we have a segment of the class  $C\_Agent$  ( $C_a$ ) as follows:

```
class C_Agent
{
  //...
  C_Role current;
  Set qualified;
  public C_Agent()//a constructor of C_Agent
  { //...
    qualified = new HashSet();
    current = null;
    //...
  }
  //...
}
```

From the code segment, we know that a new agent is created with no the current role and qualified roles. It is required to execute  $play(C\_Role)$  to attach roles.

## V. AGENT COLLABORATION BASED ON ROLES

The autonomous property and the cooperative property sometimes are a couple of contradictions for an agent. To behave well in a team work, a robot agent must make decisions based on the incoming messages and the sensor events obtained by its own sensors. To respond to incoming messages is cooperative and to behave based on its sensor events is autonomous.

Thanks to role mechanisms, pre-defined processing in relevant roles specifies how an agent is to cooperate. To

collaborate with other agents in a team, an agent sends messages to other agents and accepts messages based on its current roles. As long as we separate roles into a fine enough grain, a role can be used to express an agent's autonomous behavior or its cooperative behavior. We can concentrate on the fact that the current role played by an agent confines the messages it can accept and send out. Therefore, a message or a sensor event may make an agent transfer its current role. The major cooperative strategy is reflected by the role transfer policy.

For example, we can separate the common forward role into a serving forward role and an attacking forward. If a robot agent currently playing a defender role takes the ball from its opponent. It transfers its current role to an attacking forward role and would prefer to move the ball directly to attack the opponent's gate. If at the same time, another agent currently playing an attacking forward sends a message to it and ask it to transfer the ball, it will check its sensor events and decide what role it needs to transfer. If the situation is better for it to attack than to serve, it will transfer to an attacking forward role. If the situation is better to serve, it will transfer to a serving forward. That is to say, whether to be autonomous or cooperative is up to the situation. The result is a role transfer.

The above strategy is reasonable because in a robot soccer game the distance between an agent and the ball, the distance between two agents, and the speed of an agent and the speed of a ball are measurable and computable. The agent can behave based on these computations.

## VI. TEAM DESIGN

From the above discussed E-CARGO model, we can construct a role net to express the role relationships among a cooperative structure. There might be request/serve relationships in short-term collaboration such as a meeting and a soccer game, promotion relationships in a long-term collaboration such as an organization's development, and conflict relationships in both short-term and long-term collaborations.

The major tasks for robot team designers are designing and clearly describing the behaviors of roles and describing the interrelationships among the roles. A robot should be able to store the descriptions of roles. Therefore, role-based approaches in robot team work are mainly the process and relationship descriptions of roles.

For a robot, a role is a wrapper. With this wrapper, the robot concentrates on special tasks. It does not respond to some events or messages it actually obtained based on its current role. For example, when a robot is playing a forward role, it will go fast towards a ball and move the ball towards the gate. When he needs to block the ball, he needs to play the defender role at first. That is to say, when a robot is playing a specific role, it cannot respond to some messages or events relevant to other roles temporarily.

Agent designing is a complex task. Designers need to simulate intelligent procedures of people in an agent.

Searching and retrieving are a way to express intelligence. Roles can be used to simplify the complexity of agent design, because roles separate concerns of an agent into different aspects. Based on the theory of searching and retrieving, this separation greatly shrinks the space for searching and will significantly increasing the efficiency for agents to respond to messages or events relevant to a specific role.

In E-CARGO, every group has an environment. An environment confines a group of agents, i.e., robots. Team designers mainly consider the structure of roles, i.e., the role net (Fig. 4).

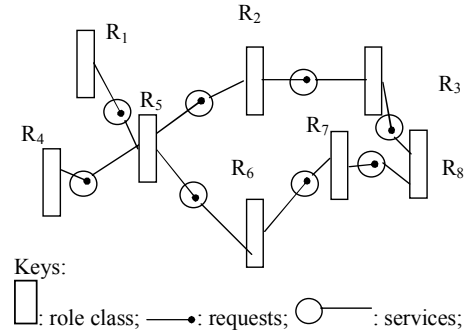


Fig. 4. A role net.

In this role net, when an agent plays a role, it actually takes a role as its working context. It accepts incoming messages and issues outgoing messages within the context of his current role. This helps the agent use all its power to concentrate on the work specified by the current role.

With the revised E-CARGO model, a role net is actually an environment. We can define a 4-4-2 style soccer team as an environment as

- $e1 = \{<defender, [4], f>, <midfield, [4], f>, <forward, [2], f>, <goalie, [1], f>\}$ , where,  $f = \{the\ gate, the\ field, the\ ball\}$ ;

A 3-6-1 style team as

- $e2 = \{<defender, [3], f>, <midfield, [6], f>, <forward, [1], f>, <goalie, [1], f>\}$ ;

If we specify the team in a more clear way, the environments are:

- $e1 = \{<left-defender, [1], f>, <right-defender, [1], f>, <mid-defender, [2], f>, <left-midfield, [1], f>, <right-midfield, [1], f>, <mid-midfield, [2], f>, <left-forward, [1], f>, <right-forward, [1], f>, <goalie, [1], f>\}$ .
- $e2 = \{<left-defender, [1], f>, <right-defender, [1], f>, <mid-defender, [1], f>, <left-midfield, [2], f>, <right-midfield, [2], f>, <mid-midfield, [2], f>, <left-forward, [1], f>, <goalie, [1], f>\}$ .

The environment  $e1$  can be shown as in Fig.5, where,  $R_1-R_4$  are defender role instances,  $R_5-R_8$  are midfielder role instances,  $R_9-R_{10}$  are forward role instances, and  $R_0$  is the goalie role instance.

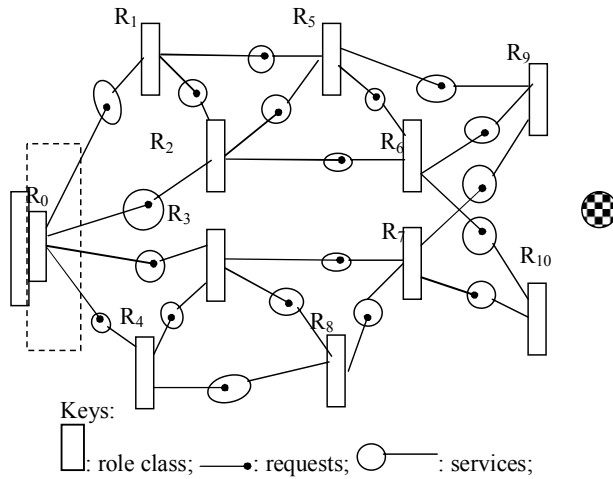


Fig. 5. Role net: an environment for a soccer team with 4-4-2 formation.

To organize a robot soccer team specified by Fig. 5, we can have 11 robots and assign them role instances to form a group  $g$  (Fig. 6). Role transfers are triggered with events activated by the sensors and the conditions predefined. Fig. 6 expresses that this formation is dynamic. It is initially 4-4-2. When attacking, it might be 2-4-4. When blocking, it might be 4-6-0. This completely supports the dynamic role allocations for agent that are required in a robot agent team [3, 6, 7].

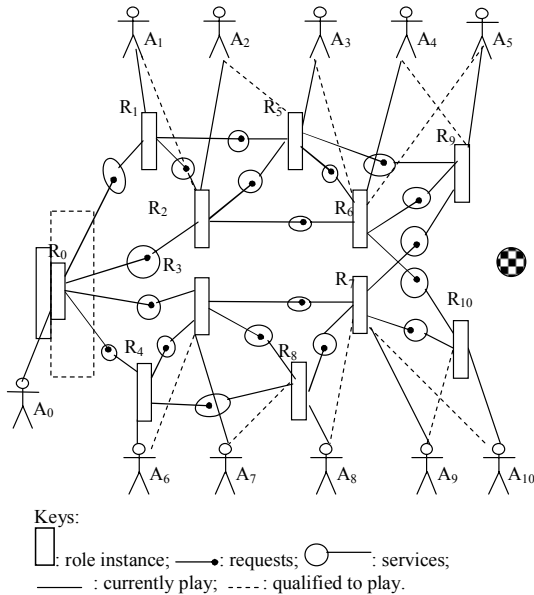


Fig. 6. Role net: a robot soccer team

Based on the definitions of group and environment, we can write classes  $C\_Environment$  and  $C\_Group$  with the supplement classes  $RoleRange$  and  $RoleMap$ .

## VII. CONCLUSIONS AND FUTURE WORK

Agent design and agent team design are complex tasks. A role-based approach may simplify the design of robot agent teams. We have discussed the design of different components

for an agent team. All the above discussions demonstrate that our revised E-CARGO is a good and practical model for role specification, agent design and team design and it can be used to simplify the design of an agent team by separating different concerns.

There are still many topics that need more research and practice. For example, a role instance is unique for an agent. The problem is that in an environment, there is a limit for the number of the role instances of the same role class, i.e., the  $q$  in  $e$ . For example,  $q = [2, 5]$  means that the largest number of role instances is 5. This will lead to so-called role competition problems. This needs further investigation. Other questions that still need to be investigated include: What roles would be really practical for a robot soccer team? How to describe the behavior of each role? When is the best time for an agent to transfer roles?

## REFERENCES

- [1] Cabri, G., Ferrari, L. and Leonardi, L., "Injecting roles in Java Agents through Runtime Bytecode Manipulation", *IBM Systems Journal*, vol. 44, no. 1, 2005, pp. 185-208.
- [2] Castelfranchi, C., "Modeling Social Action for AI Agents", *Artificial Intelligence*, vol. 103, 1998, pp. 157-182.
- [3] Chaimowicz, L., Campos, M. F. M. and Kumar, R.V., "Dynamic Role Assignment for Cooperative Robots", *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA'02)*, Washington, DC, May 2002, pp. 293-298.
- [4] Franklin, S. and Graesser, A., "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", *Proc. of the 3rd International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag, 1996.
- [5] Gruver, W., "Technologies and Applications of Distributed Intelligent Systems", *IEEE MTT-Chapter Presentation*, Waterloo, Canada, 2004.
- [6] Hayes-Roth, B., "An Architecture for Adaptive Intelligent Systems", *Artificial Intelligence*, vol. 72, no. 1-2, Jan. 1995, pp. 329 - 365.
- [7] Stone, P. and Veloso, M., "Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork", *Artificial Intelligence*, vol. 110, 1999, pp. 241-273.
- [8] Wooldridge, M. and Jennings, N. "Intelligent Agents: Theory and Practice", *The Knowledge Engineering Review*, vol. 10, no. 2, 1995, pp. 115-152.
- [9] Zhu, H. "Encourage Contributions by Roles", *IEEE International Conference on Systems, Man and Cybernetics*, Oct. 2005, Hawaii, USA, pp. 1574-1579.
- [10] Zhu, H. "Role Mechanisms in Collaborative Systems", *International Journal of Production Research*, vol. 41, no. 1, Jan. 2006, pp. 181-193.
- [11] Zhu, H. and Zhou, M.C., "Role-Based Collaboration and its Kernel Mechanisms", *IEEE Trans. on Systems, Man and Cybernetics, Part C*, to appear, 2006.