Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

Script-based Architecture for Multi-agent Robotic Systems

Fang-Chang Lin and Jane Yung-jen Hsu
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan, R. O. C.
fclin@robot.csie.ntu.edu.tw yjhsu@csie.ntu.edu.tw

Abstract

This paper proposes a generic framework for developing multi-agent systems. Various agent models or interaction mechanisms can be systematically implemented and evaluated. A script specifies a multi-agent system by defining each agent as a Finite State Automata (FSA). By using the state transition rule as the building block, an agent as well as a multi-agent system can be developed efficiently. A set of rules composes a protocol set which represents some agent's behavior. Various protocol sets can be developed for different agent models. The framework is further applied to multi-agent robotic systems in which reactivity and interaction are more important issues. Hence, each protocol set is decomposed into several module sets by functionality. Advantages of the decomposition are modulization, reactivity, parallel performance, and module set sharing. In addition, multiple protocol sets can be implemented for dynamic adaptation. A cooperation protocol for the Object-Sorting Task is utilized to illustrate the framework.

Keywords: dirtributed AI, robotics, agent models and architectures, multi-agent robotic systems

1. Introduction

Multi-agent robotic systems employ multiple robots to deal with tasks and to solve problems. One motivation of using multiple agents is for parallel performance. Some tasks such as cleaning, painting, and search, can be decomposed into several independent parts. Multiple agents can perform the tasks simultaneously. On the other hand, some other tasks cannot be achieved by a single agent, e.g. conferences, moving a heavy equipment, or formation. Most multi-agent tasks require cooperation among agents through coordinating the actions of each agent. Situations such as forage and object retrieval, movement of large equipment, information search in distributed servers, and virtual conferences are typical application domains in which explicitly cooperation is required to accomplish the tasks. If agents are self-centered and work independently, the task achievement would be accidental. Hence, there should be protocols (via negotiation, coordination, or social rules) among agents such that agents coordinate their behaviors to reach their goals as well as a cooperation architecture within which the protocols can be implemented effectively and efficiently.

There are many approaches concerning the interactions among agents. Some research models agents based on the architecture of actions, beliefs, and desires [5]. Negotiation [10,13], contracts [15], planning [4], or social rules [7,8,14] are employed for solving conflicts or inconsistency. The agents in distributed sensor network [6] communicate local plans and eventually converge their solutions. Each of the approaches is under agent model which can be rational, cooperative, benevolent, self-interested, or rule-abiding.

Regardless of the agent model adopted, we need a good methodology to systematically develop multi-agent systems and test the underlying interactions. This paper proposes a generic multi-agent framework that has the following characteristics:

- *Flexibility*: Various models or assumptions for agents can be implemented. In addition, various interaction mechanisms can be developed systematically.
- *Conciseness*: A framework should be concise. For example, the description of world can be reduced to the description of *an* agent rather than the

description of *all* the agents if the agents are homogeneous.

- *Realization*: Any interaction mechanism should can be realized. An simulation environment is provided for implementation, experiment and evaluation.

When the framework is applied to the domain of multiple robots, several issues should be addressed:

- **Reactivity**: In a dynamic environment, each agent interacts with its environment including other agents. Hence, it is impossible to deliberate every actions in real-time. Reactive behavior is necessary for quick response to the environment.
- **Interaction**: Due to the uncertainty of sensors, the changing environment, and the complex interactions, it is impractical to pre-plan a complete sequence of actions in advance. In addition, the world model may not be sufficient for planning. Rather than trying to find a complete plan that may fail, it is more important to focus on the interactions between an agent and its environment or other agents.
- **Availability**: The robots are distributed in the work space. Any movement between two locations takes time. Hence, any algorithm or mechanism must consider this issue rather than assuming the agents are always available.
- **Interference**: Since robots are physical entities, they are more likely to interfere with one another when they are close together than far away.
- **Achievement**: Although we would like a robot to be autonomous, performing tasks specified by human are the most important mission of the robot. It is necessary to have an interface for a human supervisor to assign tasks, monitor and control task execution, as well as for the robot to request help from the supervisor.

In general, a centrally controlled robot architecture cannot satisfy all the above requirements. An architecture consisting of several control levels should be used for multi-agent robotic systems. The *reactivity* level contains basic capabilities reacting with the environment by controlling physical units. The *interaction* level includes functional modules performing interaction mechanisms by controlling the reactivity level. The *interface* level interacts with the human supervisor, manages interaction mechanisms, and controls functional modules. All *active* modules or capabilities can execute simultaneously.

Reactive control is motivated from the stimulate-reaction mechanism studied in ethology and sociology. Many behavior-based architectures have been utilized in the robot control because of its reactivity, robustness, and flexibility. Brooks [2] proposed and utilized the subsumption architecture to control their robots on applications. Arkin [1] motivated from schema theory and proposed schema-based navigation for reactive robot systems. These architectures were demonstrated only on basic behaviors such as avoidance, escape, and navigation, etc. To achieve a goal requiring planning or world modeling, some research intended to incorporate deliberate planing and reactive control. Connell [3] proposed a hybrid architecture combining a servo-control layer, a subsumption layer, and a symbolic layer. The architecture was illustrated by an indoor navigation example. Watanabe et at. [16] used an adaptive level on the behavior-based architecture to recover from failure during executing a sequence of motions.

Some other research proposed the architecture for multi-agent robotic systems. Noreils [11] proposed a cooperative architecture for mobile robots, which focused on task decomposition. Agents negotiated for decomposing a global task and resolved conflicts in their plans represented by a sequence of actions, then each agent executed its plan. Pape [12] utilized a central task planner and scheduler for multiple robot coordination. Individual robots determined their contribution for a global task, then the central planner decided the final task allocation. Since deliberate planning and reasoning are embedded into the complex architectures, the reactivity of agents decreases. Furthermore, most of the proposed architectures focus on task decomposition and negotiation for task assignment. Nevertheless, the reactivity of performing tasks is an important issue to exploit.

The following section introduces the *Object-Sorting Task* (OST) which will be utilized to illustrate the proposed framework in the other sections. Next, the framework are described. Finally, the help-based cooperation protocol of the OST is demonstrated for the framework.

2. The Object-Sorting Task (OST)

Let $O=\{o_1,...,o_M\}$ be a set of stationary objects that is randomly distributed in a bounded area. Every object, $o_i=(l_i,d_i,n_i)$, is associated with an initial location l_i, a destination location d_i, and the number n_i of agents for movement. An object o_i can be moved if and only if there are at least n_i agents available to move it. Let $R=\{r_1,...r_N\}$ be the set of agents and n_{max} be the maximal number of agents to move any single object, an object-sorting task can be completed only if N is not less than n_{max}. Agents search for objects and move them to their destinations. When all the objects have been moved to their destinations, the task is finished.

For the OST, the functional modules can be *search*, *movement*, and *cooperation*. The search module searches for objects. The movement module moves objects alone

or together with the other agents. The cooperation module exchanges messages with the other agents for cooperation.

A formal definition of the OST as well as a genetic algorithm for approximating the optimal solutions can be found in [9]. Two cooperation protocols for the OST were developed in [7,8].

3. The framework

A multi-agent system includes the agents, the *environment*, and their *interactions*. This paper proposes an approach to design a multi-agent system based on the design of individual agent. Starting from the design of each agent as well as its interactions with the environment and other agents, a multi-agent system is created by combining the involved agents.

A *script* specifies a kind of multi-agent system in which one or more interaction mechanisms are emp'oyed. Implementations of diverse interaction mechanisms result in different scripts. For example, a system may use negotiation in self-interested agents, or employ social rules in cooperative agents, etc. Since there are a wide variety of multi-agent systems, it is useful to have a general framework in which different scripts can be developed, implemented, and evaluated.

First, we need a representation from which various scripts can be generated. Each agent is modeled by an extended *Finite State Automaton* (FSA). Furthermore, it is modularized through function and capability. Finally, an architecture is needed to realize the model.

3.1 Script representation

In the world of multi-agent systems, each agent performs its own task as well as interacts with the environment and the other agents. Status or change of the world may be sensed by an agent. The sensed result is an input event to the agent and may effect the internal state of the agent. An a result, the agent may take some actions reacting to the event and change its internal state. Hence, the concept of FSA is extended to model an agent.

Definition. An agent is $R=(S, E, A, T)$:

- S is a finite set of the agent's states. In each state, the agent may perform some tasks as well as handle input events.
- E is a set of input events. An event may be:
  - ☐ a result sensed: e.g. finding a large object
  - ☐ a message received: e.g. a help message received
  - ☐ a condition satisfied: e.g. staying in a state more than a specified duration (timeout)
- A is a set of actions. An action may:
  - ☐ control physical units: e.g. issuing a motor command
  - ☐ send messages: e.g. broadcasting a help message
  - ☐ make decisions: e.g. choosing the nearest partner from a group of candidates and sending a message to it
- T is a function, T: $S\times E \rightarrow S\times A$, means events cause actions taken and state change. It is possible that the state keeps no change or no action taken. A transition is called a *rule* represented by the following definition.

Definition. A rule $f=(s, e, a, t)$ is a 4-tuple where

$s \in S$: the agent's current state
$e \in E$: an event
$a \in A$: an action
$t \in S$: the next state

Definition. A script $P=\{R_1, R_2, ..., R_n\}$ defines n agents, where R_i is the i-th agent.

Every rule defines what actions an agent should take when events occur. All the rules of an agent specified in a script is a *protocol set*. Different protocol sets may be utilized for agents to manage different situations. A protocol set is further decomposed into several *module sets*, each module set is associated with a functional module and defines the module's behavior.

3.2 Agent architecture

The architecture of each agent consists of *physical units*, *basic capabilities*, *functional modules*, and the *protocol manager*, as depicted in Fig. 1. The protocol manager interacts with human, governs all protocol sets, and controls the behavior of each functional module by changing the module rules of each module. It adds, deletes, performs protocol sets under the control of an outside supervisor which may be a human or another agent.

A protocol set defines the transition of a FSA, i.e. the behavior of an agent. The FSA is realized by several functional modules which perform functions in parallel. They share the global state information and change the state information according to their *partial* state transition functions. A module's transition function is represented by a working module set which defines the underlining behavior of the module.

When an agent starts a specified protocol, the protocol manager will retrieve the underlining protocol set *cp* and load its module sets ($m_1, m_2, ..., m_h$) for each associated functional module ($FM_1, FM_2, ..., FM_h$). Then the manager activates the h functional modules. Under this

situation, the protocol set *cp*, the *h* functional modules, and the *h* module sets are all *active*.

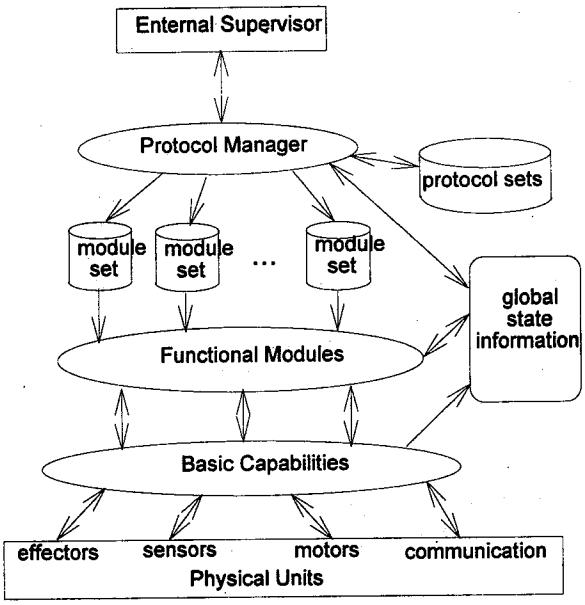

Fig. 1: Generic agent architecture

Each functional module controls several basic capabilities of the agent. Reactive behavior-based design is utilized for the basic capabilities. Each basic capability reacts with the environment by managing the related physical units such as end effectors, sensors, motors, and communication systems.

An event may come from the basic capabilities or the functional modules. For example, if the *object detection* behavior finds a large object, it will issue a "large object found" event. When the communication system receive a *help* message, it issues a "received help" event. When an agent have waited for a specified duration, the motion module issues an "timeout" event.

The advantages of the proposed architecture are summarized as follows:

- The architecture is reactive and modularized in low level basic behaviors.
- Each module performs a part of the global FSA, i.e. only focus on the rules in its active module set rather than the global FSA. And modules can execute simultaneously. Hence, the architecture is reactive and modularized in functional layer.
- Both rules and module sets can be shared by different protocol sets.
- A cooperation protocols can be represented by a set of cooperation rules.
- Multiple protocols can be implemented into the architecture. Moreover, different protocol sets can be activated or inactivated dynamically in changing environment.

4. Demonstration

The *help-based cooperation protocol* [7] for the *Object-Sorting Task* is illustrated to demonstrate the framework. Several assumptions were made in our research. The agents are *homogeneous* mobile robots with the basic capabilities for *navigation*, *obstacle avoidance*, *object recognition*, and *object handling*. The agents are cooperative and follow *cooperation protocols*. Finally, the communication system is *reliable*, and the agents communicate with the others by broadcast or point-to-point messages.

4.1 Problems of the OST

The OST consists of two kinds of work. One is the *search* work: searching for objects, another is the *object processing* work: carrying objects to destinations. The former has search problem while the latter has coordination and deadlocks problems. To solve the OST, these problems need to be addressed .

1. *Search*. All the objects must be found in order to attack the task. How do they search efficiently ?
2. *Coordination*. How do the agents coordinate for assigning themselves to a found object ? That is, for a given object, which agents should work together to move it ? and who makes the decision ?
3. *Deadlocks*. When each agent autonomously selects an object and none of the selected object has enough agents for movement, a deadlock occurs.

4.2 Help-based script

The working area is equally partitioned into disjoint *N* subareas, and each subarea is assigned to an agent. Advantages of the equally partition are load balancing and parallel performance of the *search* work. Each agent exhaustively searches its subarea, requests help from the others once it has found a large object, and selects its partners. After there are enough agents arriving at the found object, they carry the object to its destination. The cooperation protocol defines how and when an agent requests help, how and when the other agents offer help, and deadlock handling in order to coordinate the agents for accomplishing the task.

Three functional modules are required: the *search* module, the *motion* module and the *cooperation* module. The search module searches and identifies objects. The motion module performs navigation and object movement (alone or coordinated with other agents). The cooperation module cooperates with the other agents by communication messages. Fig. 2 shows the agent architecture designed for the OST.

The help-based script defines the protocol set of each agent. The state diagram for each agent is showed in Fig. 3. The rules is presented in the following subsection. Furthermore, the protocol set is functionally decomposed into three module sets which are handled by each module. Fig. 4 shows the transition of each module.

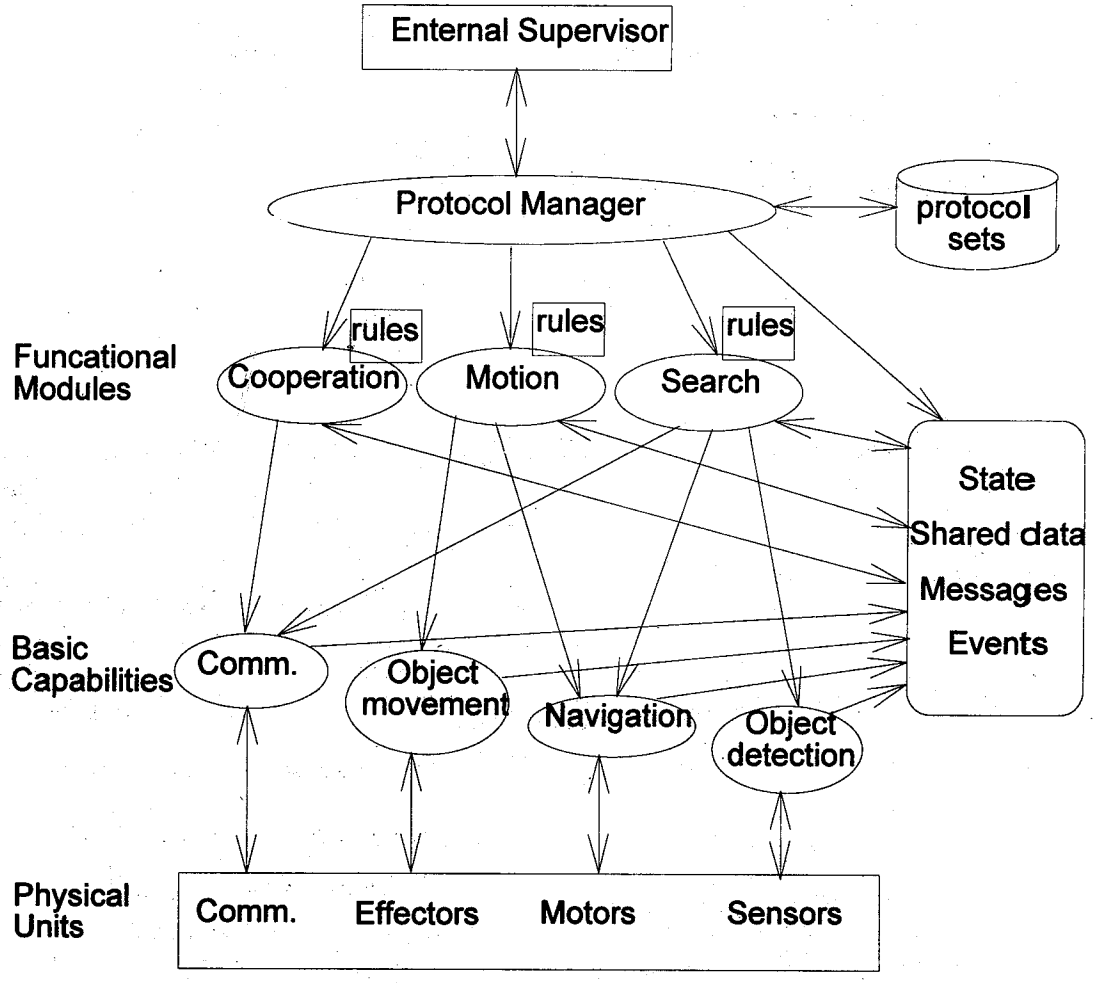

Fig. 2: Agent Architecture for the Object-Sorting Task

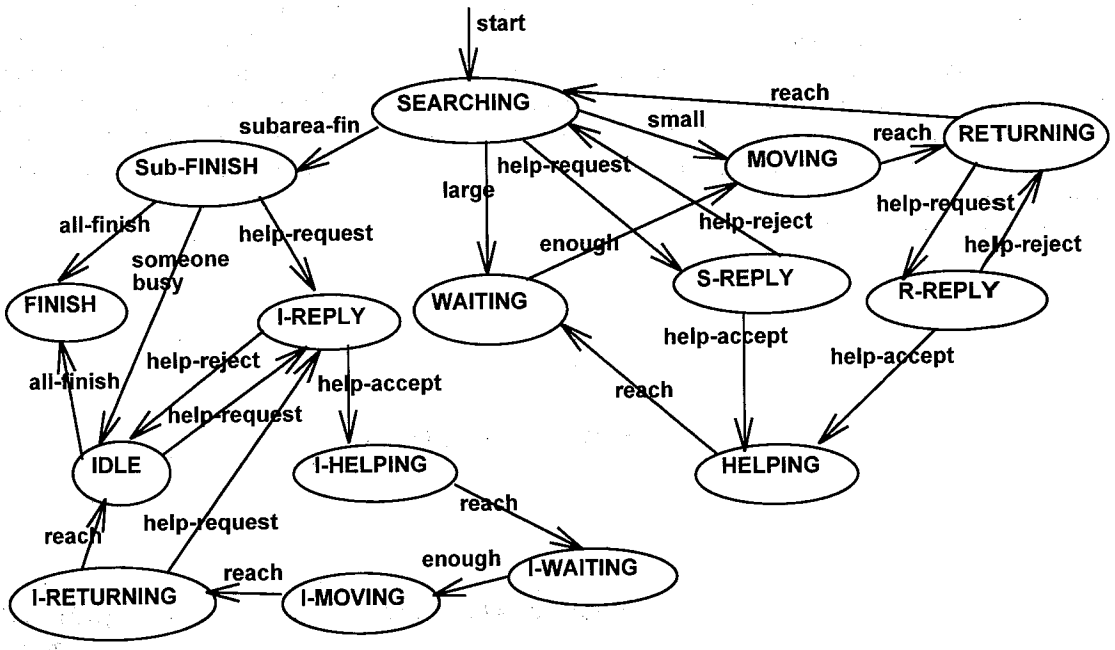

Fig. 3: State transition diagram

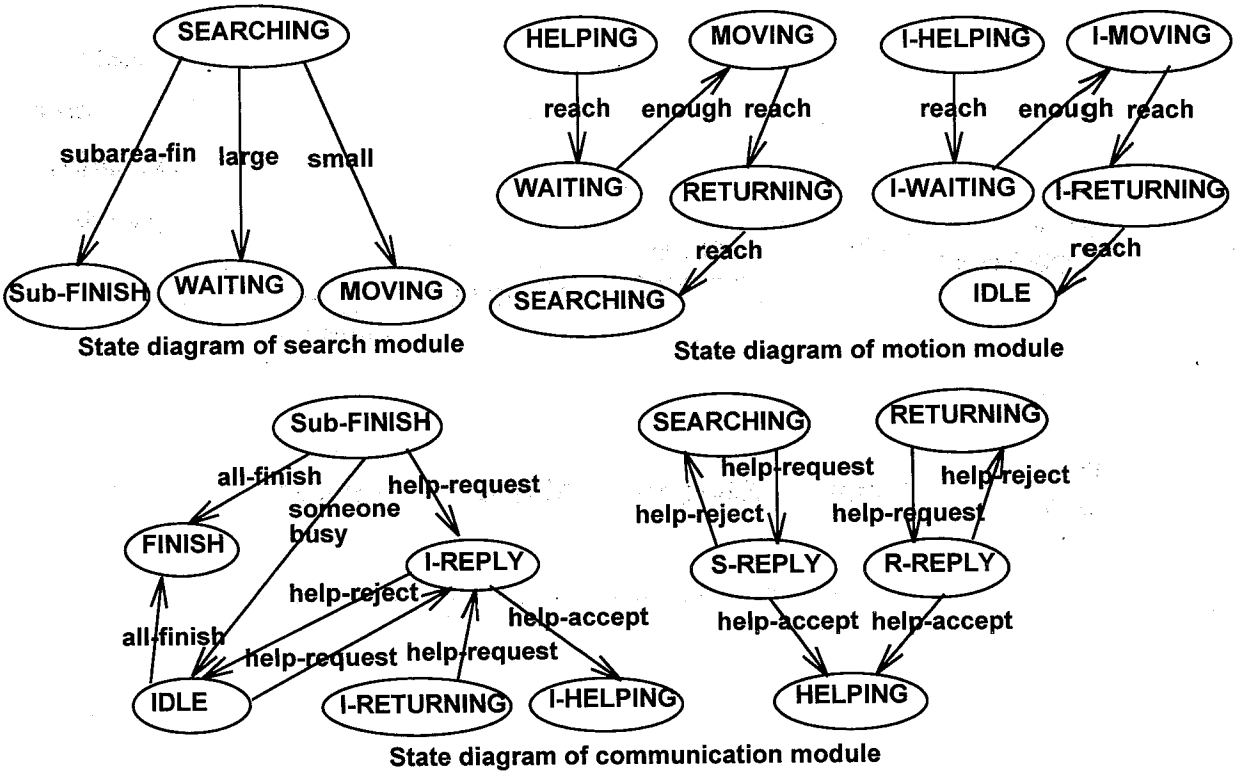

Fig. 4: State diagrams of the modules

4.3 The protocol set

Several strategies are used to illustrate the protocol. The *request-help* strategy specifies when an agent request help from the others. The *offer-help* strategy defines when to offer help. The *select-help* strategy determines an agent to offer help. The select-partner strategy selects the partners to work together. The *deadlock-free* strategy is to detect or prevent the system from deadlocks. The following notations are used for abbreviation.

D-*s*: a sensed event
R-*m*: a message received
C-*c*: a condition satisfied
S-*m*: to send a message
M-*d*: to make decisions
"-": empty action or no state change

Request-help:

When an agent finds a large object, it sends a *help* message to request help and enters the WAITING state.

(SEARCHING, D-*large-object*, S-*help*, WAITING)

Offer-help:

Let AVAIL be the set of these states {SEARCHING, RETURNING, I-RETURNING, SUB-FIN, IDLE}, ~AVAIL represent the states not in AVAIL, and R-AVAIL={S-REPLY, R-REPLY, I-REPLY}. When an agent in AVAIL and there exist any help requests, it selects a request based on the *offer-help* strategy and replies it. Then it enters one of the reply states. After that, when it receives an *accept* message, it enters HELPING state. On the other hand, it returns to its previous state when it receives a *reject* message.

(SEARCHING, R-*help*, M-*offer-help*, S-REPLY)
(RETURNING, R-*help*, M-*offer-help*, R-REPLY)
(I-RETURNING, R-*help*, M-*offer-help*), I-REPLY)
(SUB-FIN, R-*help*, M-*offer-help*, I-REPLY)
(IDLE, R-*help*, M-*offer-help*, I-REPLY)
(~AVAIL, R-*help*, enqueue *help*, -)
(R-AVAIL, R-*accept*, - , HELPING)
(S-REPLY, R-*reject*, - , SEARCHING)
(R-REPLY, R-*reject*, - , RETURNING)
(I-REPLY, R-*reject*, - , IDLE)

"M-*offer-help*" is to perform the *offer-help* strategy. For example,

M-*offer-help*: Send a *reply* message to the nearest agent among the received help-requests.

Select-partner:

When an agent in WAITING state and there exist any received *reply* messages, it select its partners by the *select-partner* strategy.

(WAITING, R-*reply*, M-*select-partner*, -)

M-*select-partner*: Let n be the required number of agents, i be the current number of selected partners, and h be the number of reply messages. S-*accept* to the nearest n-i agents and S-*reject* to the other h-(n-i) agents if $h > n$-i. Otherwise, S-*accept* to the h agents.

Deadlock-free:

When an agent stays in WAITING for a duration of the "wait timeout", it will broadcast an *isblocked* message and enter the BLOCKED state. The other agents not in WAITING will reply a *notblocked* message if it receives an *isblocked* message. If an agent stays in BLOCKED for the duration of "blocked timeout", the agent will broadcast a *blocked* message to state that the system is deadlocked. The agents will use the specified deadlock resolution to break the deadlocked state. One of the resolution states that the agents will help the highest priority agent to move its object.

(WAITING, C-*wait-timeout*, S-*isblocked*, BLOCKED)
(~WAITING, R-*isblocked*, S-*notblocked*, -)
(BLOCKED, R-*notblocked*, - , WAITING)
(BLOCKED, C-*blocked-timeout*, S-*blocked*, -)
(BLOCKED, R-*blocked*, M-*resolution*, -)
(BLOCKED, R-*accept*, - , HELPING)
(BLOCKED, R-*reject*, - , WAITING)

M-*resolution*: If I am the highest priority agent, keep state in WAITING, else S-*reply* to the highest priority agent.

5. Conclusion

In this paper, a generic framework for designing multi-agent robotic system is proposed. A *script* specifies a kind of multi-agent system by defining each agent's behavior as well as its interaction with the environment and other agents. An agent is modeled by the concept of *Finite State Automata*. A reactive and modularized architecture is utilized to realize the agent model for multi-agent robotic systems. In addition, the *Object-Sorting Task* is employed to illustrate the framework.

By the representation of transition *rules*, a protocol set can be decomposed into several module sets so that they can perform in parallel. Furthermore, a module set can be shared by different protocol sets. Advantages of the agent architecture are summarized as follows:

- The architecture is reactive, modularized.
- Each module performs a part of the global FSA. And all modules can execute simultaneously.
- A cooperation protocols can be represented by a set of cooperation rules.
- Multiple protocols can be implemented into the architecture. Moreover, different protocol sets can be activated or inactivated dynamically.

The simulation environment described in [7] provides an experiment environment for the proposed framework, and can be modified to generalize the environment. In addition, a language can be developed for the specification of protocol sets.

References

[1] R.C. Arkin, "Motor Schema-Based Mobile Robot Navigation", *International Journal of Robotics Research*, 8(4):92-112, 1989.

[2] R.A. Brooks, "A Robust Layered Control System For A Mobile Robot", *IEEE Journal of Robotics and Automation*, RA-2(1):14-23, 1986.

[3] J.H. Connell, "SSS: A Hybrid Architecture Applied to Robot Navigation", *Proc. of the 1992 IEEE International Conference on Robotics and Automation*, Nice, France, May 1992, pp. 2719-2724.

[4] E.H. Durfee, and V. R. Lesser, "Using Partial Global Plans to Coordinate Distributed Problem Solvers", Alan H. Bond and Les Gasser (Eds.), *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., 1988.

[5] M.R. Genesereth, M.L. Ginsberg, and J.S. Rosenchein, "Cooperation Without Communication", *Proc. of the National Conference on Artificial Intelligence*, 1986, pp. 51-57.

[6] V.R. Lesser and D. D. Corkill, "The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks", Robert Engelmore and Tony Morgan (Eds.), *Blackboard Systems*, Addison-Wesley Publishing Company, 1988.

[7] F.C. Lin and J.Y.-j. Hsu, "Cooperation and Deadlock-Handling for an Object-Sorting Task in a Multi-agent Robotic System", *Proc. of the IEEE International Conference on Robotics and Automation*, Nagoya, Japan, May 1995, pp. 2580-2585.

[8] F.C. Lin and J.Y.-j. Hsu, "Coordination-based Cooperation Protocols in a Multi-agent Robotic System", *Proc. of the IEEE International Conference on Robotics and Automation*, Minneapolis, Minnesota, April 1996.

[9] F.C. Lin and J.Y.-j. Hsu, "A Genetic Algorithm Approach for the Object-Sorting Task Problem", *Proc. of the IEEE International Conference on Systems, Man, and cybernetics*, Vancouver, Canada, Oct. 1995.

[10] K. Matsubayashi and M. Tokoro, "A Collaboration Mechanism on Positive Interactions in Multi-agent Environments", *Proc. of IJCAI 93*, 1993, pp. 346-351.

[11] F.R. Noreils, "An Architecture for Cooperative and Autonomous Mobile Robots", *Proc. of the 1992 IEEE International Conference on Robotics and Automation*, Nice, France, May 1992, pp. 2703-2710.

[12] C. Le Pape, "A Combination of Centralized and Distributed Methods for Multi-Agent Planning and Scheduling", *Proc. of the IEEE International Conference on Robotics and Automation*, 1990, pp. 488-493.

[13] J.S. Rosenschein and M. R. Genesereth, "Deals Among Rational Agents", *Proc. of IJCAI 85*, Los Angles, California, Aug. 1985, pp. 91-99.

[14] Y. Shoham and M. Tennenholtz, "On the Synthesis of Useful Social Laws for Artificial Agent Societies (Preliminary Report)", *Proc. of AAAI 92*, 1992, pp. 276-281.

[15] R.G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver", *IEEE Transactions on Computers*, Vol. C-29, No. 12, Dec. 1980, pp.1104-1113.

[16] M. Watanabe, K. Onoguchi, I. Kweon, and Y. Kuno, "Architecture of Behavior-based Mobile Robot in Dynamic Environment", *Proc. of the 1992 IEEE International Conference on Robotics and Automation*, Nice, France, May 1992, pp. 2711-2718.