Behavioral Simulation Based on Knowledge Objects

Takeo Maruichi, Tetsuya Uchiki, and Mario Tokoro
Department of Electrical Engineering
Keio University
3-14-1 Hiyoshi, Yokohama 223
JAPAN
+81 44 63 1926
maruichi%keio.junet@japan.cs.net
mario%keio.junet@japan.cs.net

Abstract

The purpose of behavioral simulation is to simulate the behavior of the characters having behavior rules in the surrounding environment. This kind of simulation, which differs from the traditional simulation based on a statistical model of the world, provides us with a more precise simulation with regard to individuals. Because of the nature of behavioral simulation, characters, the environment in which the characters exist, and messages which are passed among characters, are the main elements that should be considered. The notion of object-orientation is one of the most attractive computational models for providing the basis of behavioral simulation. Specifically, we employ the notion of knowledge objects which are objects having knowledge contained within themselves. We first establish a behavioral simulation model based on the notion of knowledge objects, then design and implement a behavioral simulation system PARADISE. To demonstrate the capability and effectiveness of this simulation model, *barracuda and herring school* is used as an example of this simulation system.

1 Introduction

Most simulations have traditionally been based on statistic or mathematic models about the world. However, the real world is actually made up of individuals. Therefore, if we need to simulate the world in a more precise way, simulation cannot be based on the statistics of a group's behavior, but on the rules of an individual's behavior. The goal of this research is to establish a method to build a simulated world based on the rules of the behavior of individuals. In this paper, we choose an animal's behavioral simulation as an example.

In behavioral simulation, the behavior of characters, having their behavioral rules, are simulated taking into account the surrounding environment. One good example of behavioral simulation is found in simulated motion animation such as the *barracuda and herring school* model [Partridge 1982] [Reynolds 1985], which is illustrated in Fig.1. Such kind of simulation is effective not only for animation without scripts, but also a wide variety of simulations for education and experiments, including robotic behavior, traffic control, cognition, and psychological behavior.

In order to realize behavioral simulation, we have to consider the model of a behavioral character. The notion of a knowledge object [Tokoro 1984] [Ishikawa 1987] is an attractive computational model to represent a character in behavioral simulation. A knowledge object is a concurrent object consisting of local variables, methods, and knowledge (or a set of rules). It is possible to represent a character, which decides its action based on its local state and knowledge, using the knowledge object. In order to realize behavioral simulation more exactly, the following issues should be discussed in addition:

- Representation of behavioral rules for the character

  Each character has not only actions, but also their behavior rules, which are knowledge concerning its local state and environment. A means for easily representing behavioral rules for a character must be defined.

- A simple execution mechanism for a behavioral simulation

  Basic components including a character in a simulated world should be defined in a single unified way. A simple computational model is also important.

- A programming and debugging environment for behavioral simulation

  In behavioral simulation, each character simultaneously sends/receives information (messages) to/from the environment and changes its state. A visual programming and debugging environment for such concurrent characters is inevitable.

In this paper, we investigate behavioral simulation and discuss components of a simulated world, then describe the implementation of a behavioral simulation system PARADISE. A programming example is also described.

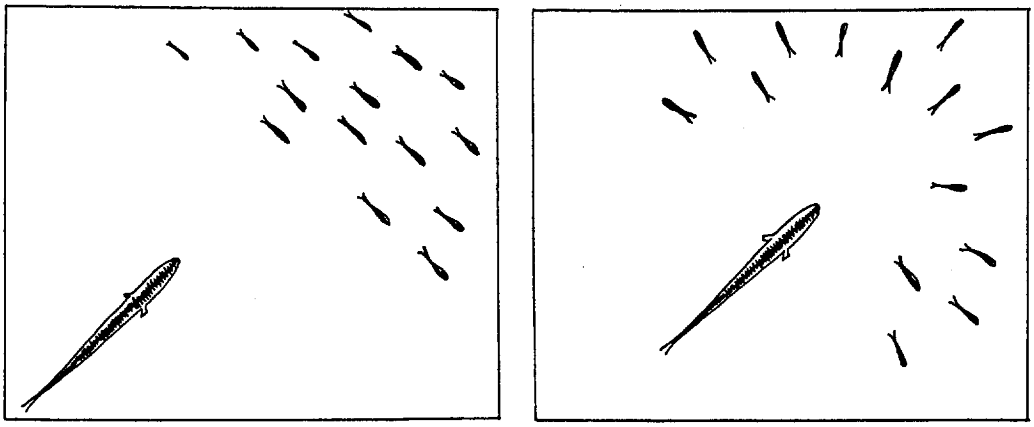


Fig.1 Barracuda and herring school

2 A Behavioral Simulation Model

In behavioral simulation, characters behave in the simulated world. We first noticed that each character is influenced by the surrounding environment. For instance, if it rains, a character's behavior differs from that in fine weather. Also, a character is a part of the environment for another character. In addition, a character might communicate with other characters. This information exchange among characters is achieved in terms of a message. In behavioral simulation, such a message is sometimes influenced by the environment. Smell is a good example. Smell actually moves with the wind. Therefore, for one character, all the other characters and messages are a part of the environment. In order to realize such a simulation world, characters, messages, and the environment are defined as follows:

2.1 Character Model

For example, a human being collects information by means of sight, sound, or smell from the environment through his sense organs, then decides its next action and behaves. A character such as a human being is an abstract model of an active entity to be simulated. This character basically behaves like a process, and collects messages voluntarily.

According to the above discussions, a character has the following functions:

1. Internal Status

   The character has its local status. This is defined by the instance variables of an object.

2. Sensor

   The sensor is similar to the eyes or ears of an animal. Each character receives information (messages) through its sensors. A sensor has its accessible area, or scope, and can receive information (messages) existing in that area of the environment.

3. Behavioral Rules

   Behavioral rules define the behavioral pattern of the character taking into account the conditions of both the character's local status and information (messages) from the environment. These rules would properly be represented by production-rules.

4. Behavioral Methods

   After defining the behavioral pattern of the character, its behavioral methods are executed, and the state of the character is changed. These methods are normally treated as local procedures.

2.2 Message Model

Many kinds of messages are used for communication in the real world, such as sight, sound, or smell. Such a message can move and change its state. Thus, a message in our simulation model is different from that in object-oriented languages. It is more natural to represent a message as an object.

A Message has the following functions:

1. Transmissive Area

   A message has its transmissive area. For example, loud sounds are transmitted far way while a quiet sound is not. This area is related to a sensor's accessible area described above.

2. Life Span

   Messages such as sounds exist only for a moment, but message such as a smell do not disappear immediately. Therefore each message has its own life span.

3. Action

   Some messages change their state. For example, a message such as a smell moves with the wind. To represent a change in state, a message has information about its location, velocity, direction, etc.

2.3 Environment Model

The environment is the world or field where characters exist. The environment contains and manages the characters and the messages. It has the following functions:

1. Management of Space

   The environment manages the distance between characters and messages. Concretely, this distance is between the accessible area of a character's sensor and the transmissive area of a message.

2. Management of Time

   All characters and messages are executed within a certain unit of time in order to maintain consistency. The environment manages synchronization among all such characters as processes. Therefore, it can be considered to have a role as the process scheduler.

3. Graphic Output

   The environment has a window to show the animation (or sequence of frames) produced by simulation. The area shown in the environment can be changed.

3 A Behavioral Simulation System : PARADISE

3.1 Simulation Mechanism

As mentioned above, each character is an active object which voluntarily searches for messages to take in from the environment. This mechanism for selecting messages is called **message searching**. Therefore, each character communicates with one other indirectly by sending and receiving messages to/from the environment. The behavioral simulation model proposed by us is illustrated in Fig.2.

A character accepts only messages, and the environment manages messages. During simulation, PARADISE's scheduler manages to calculate all the characters' behavior within one simulation unit of time, and then updates the state of the environment for the execution within the next time unit. This is repeatedly performed. The synchronization among concurrent objects is automatically performed by the PARADISE scheduler.

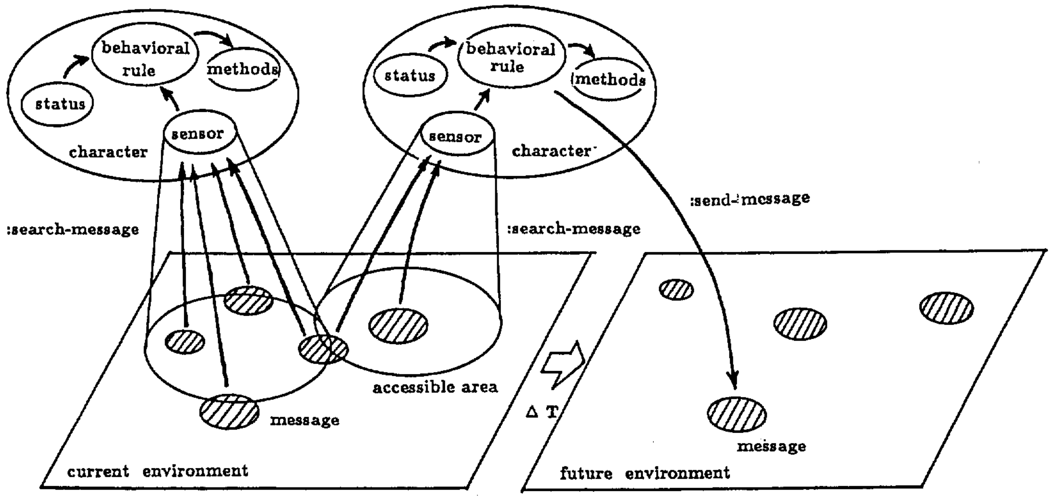


Fig.2 A behavioral simulation model

3.2 Message Management and Message Detection

The environment consists of two parts, the current message pool and future message pool. During the execution of a time unit, each character searches for messages from the current message pool and places messages into a future message pool. The procedures for accessing these two message pools are provided by the environment and sensors of a character. After all characters have finished their execution in the time unit, the environment collects all messages in the future and current message pools. The environment possesses a lifetime counter for each message and puts those into the current message pool. If its lifetime counter expires, the environment removes the message.

The sensor of a character selects messages in the environment depending on the specified ability of the sensor and the property of the messages. Though various situations for message detection can be considered, in the current implementation, only messages within the accessible area of the sensor are provided, which are shown in Fig.3.

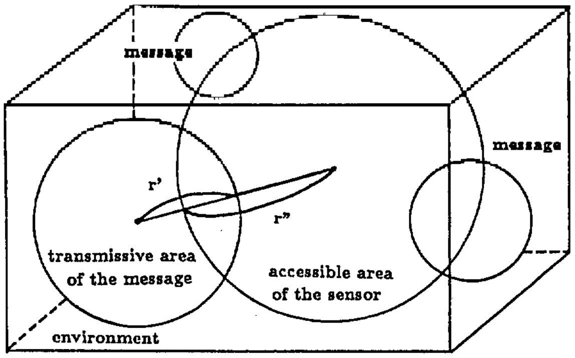

message
message
r'
r"
transmissive area of the message
accessible area of the sensor
environment

Fig.3 Message detection

3.3 Implementation

The behavioral simulation system PARADISE is implemented using PANDORA [Maruichi 1987]. PANDORA is an experimental multiparadigm programming language/environment based on object-orientation and programmed in KCL (Kyoto Common Lisp) [Yuasa 1985] running on a VAX-11 and a Sony NEWS workstation. PANDORA uses LOOPS' syntax [Bobrow 1983] and has a production system class, which is an extended OPS5 [Forgy 1981], and a process class for concurrent execution.

Characters, messages, and the environment are all designed as classes. The character class has two superclasses, one includes the production-system class and the other includes the process class. Because each character has its own production-system and executes like a process, PARADISE can be said to be a distributed production-system. Users can define their characters and messages by defining subclasses of the character and message classes, respectively.

4 Programming

The behavioral simulation of the *barracuda and herring school* model is used to exemplify how we write programs for behavioral simulation. The characters, which represent barracudas and herrings, are assumed to recognize other characters by sight messages. Therefore only sight messages are passed to/from the environment, and two kinds of characters exist in the environment.

4.1 Definition of Characters

The barracuda and herring classes are defined as characters. An example definition for herrings is shown in Fig.4. The environment, the sensor, and other variables are declared in this definition. The *environment* is an object named sea which is an instance of class environment, and has a scheduler and message pools. The sensor is a class name. When an instance of herring class is created, an instance of sensor class is also created. The variables, *position*, *direction*, and *velocity* represent the state of a herring. The variable *status* will be described in the next section. In this example, several instances of the herring class form a school and some instances of barracuda attack a school of herrings.

Each such instance is treated as a process and supervised by the PARADISE scheduler.

4.2 Behavioral Rules and Behavioral Methods

Both the barracuda and herring classes need their knowledge, or a set of behavior rules, to decide actions based on their internal status and messages collected from the environment. Such a set of rules of a character usually contains many rules, and thus programming becomes complex. To avoid this complexity, rules should be categorized by the situation of the character. For instance, a herring has the following three situations:

1. A herring is normally swimming forward when there is nothing around it.
2. A herring schools with other herrings when there is no barracuda near it.
3. A herring escapes from any barracudas when any are near it. In this case, a herring ignores other herrings because of panic.

These three conditions are named **alone**, **schooling**, and **escaping**, respectively, as shown in the state transition diagram in Fig.5. Each arrow between states is equivalent to a production rule. Each circle, which indicates a situation is equivalent to a behavioral method. Each arrow is then translated by a programmer into a production rule, shown in Fig.6.

Collected messages and a character's status are stored in the working memory of the character, and behavior rules are invoked. After behavioral rules are selected and executed, behavioral methods are invoked by message passing on the right hand side of each production rule. Behavioral methods which appeared in Fig.6 are described in Fig.7.

Then, the characters change their screened information, send new messages to the environment, and a new frame for motion animation is created.

However, in the case of more complex behavioral simulation, such as a simulation of strategies in vollyball game, it is not easy to make the state transition diagram. The effort is similar to building expert systems. This is done by understanding the actions of characters and is deeply related to behavioral psychology.

```
(defcharacter herring
        (environment      'sea)
        (sensor           ($$ sensor))
        (variables
                (pose                   'normal-right)
                (audittrail nil)
                (status                 nil)
                (position  nil)
                (direction nil)
                (velocity  nil)))
```

Fig.4 The definition of a character

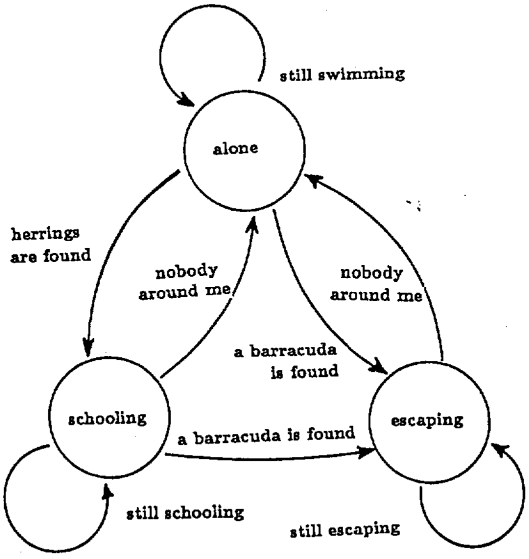


Fig.5 State transition of a herring

```
(start-production-rules herring
  (class-in-working-memory
        herring figure-message msg-list))

(p alone-swimming
        (herring ^status alone)
        (msg-list ^number-of-messages 0)
   -->
      (<- self :swimming) (halt))

(p alone-with-barracuda
        (herring ^status alone)
        (figure-message ^type barracuda)
        (msg-list ^number-of-messages <> 0)
   -->
      (<- self :@status 'escape)
      (<- self :escaping
          (<- (@wm 3) :get-character 'barracuda))
      (halt))

                  •
                  •
                  •

(p escape-still-barracuda
        (herring ^status escape)
        (figure-message ^type barracuda)
        (msg-list ^number-of-messages <> 0)
   -->
      (<- self :escaping
          (<- (@wm 3) :get-character 'barracuda))
      (halt))

(p escape-to-alone
        (herring ^status escape)
     - (figure-message ^type barracuda)
   -->
      (<- self :@status 'alone)
      (<- self :swimming) (halt))

(end-production-rules herring)
```

Fig.6 Behavioral rules of a herring

```
(defmethod swimming ((self herring))
  (prog (x y z)
        (multiple-value-setq (x y z) (<- (@ position) :get))
        (<- (@ position) :put
            (+ x (round (* (@ velocity) (cos-table (@ direction)))))
            (- y (round (* (@ velocity) (sin-table (@ direction)))))
            z)
        (multiple-value-setq (x y z) (<- (@ position) :get))
        (<- (@ sensor) :put x y z)
        (return self)))

                  •
                  •
                  •

(defmethod escaping ((self herring) barracuda)
  (prog (x y z x1 y1 z1 message r)
        (setq message (car barracuda))
        (multiple-value-setq (x y z) (<- (@ position) :get))
        (multiple-value-setq (x1 y1 z1)
                        (<- (@ (@ message sender) position) :get))
        (setq r
            (round (natural (- (* 180 (/ (atan (- y y1) (- x1 x)) pi)) 180))))
        (setf (@ direction)

            (round (natural
                    (if (and (> r (- (@ direction) 10))
                             (< r (+ (@ direction) 10)))
                        (then
                         (if (< r (@ direction))
                             (+ (@ direction) 70)
                             (- (@ direction) 70)))
                        (+ r (if (< r (@ direction)) 30 -30))))))
        (<- self :swimming)
        (return self)))
```

Fig.7 Behavioral methods of a herring

```
(defmessage      figure-message
        (direction        0)
        (type             nil))
```

Fig.8 The definition of a message

4.3 Message Passing and Message Searching

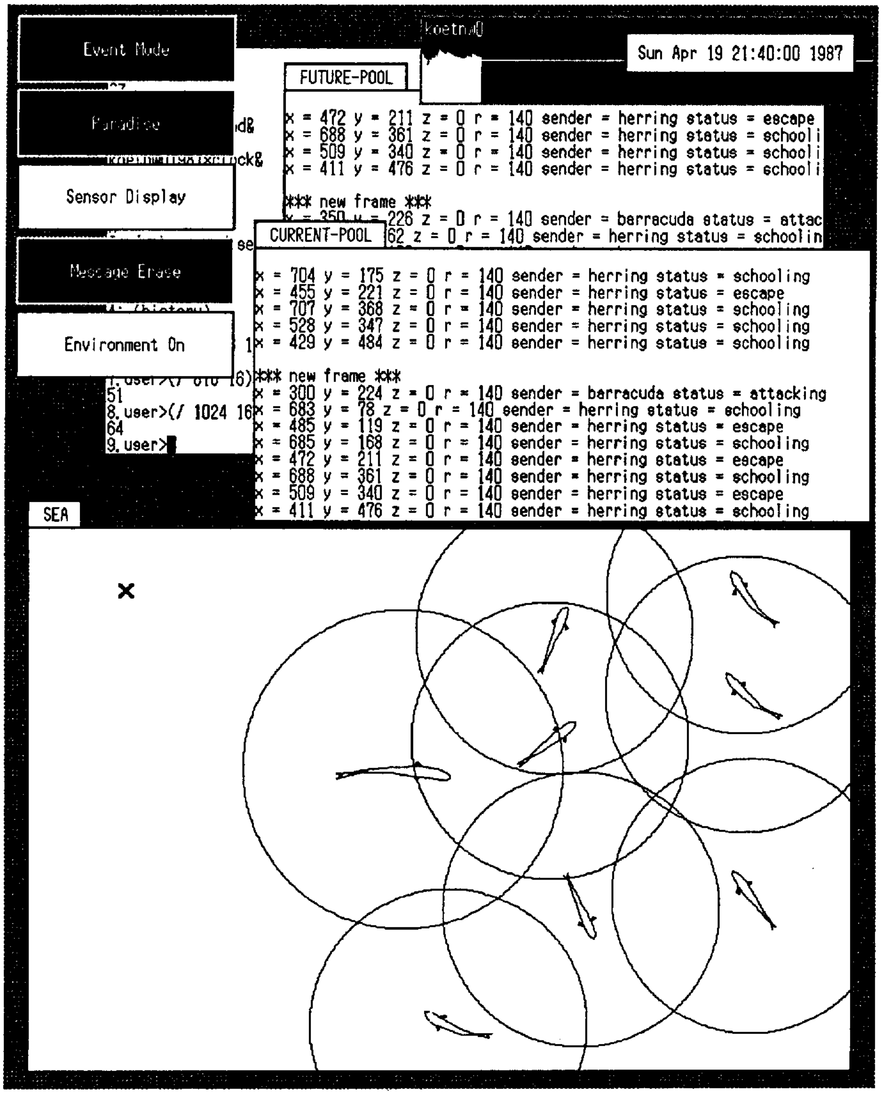
A character collects messages through its sensor(s) from the environment. All the messages are kept in the environment. For instance, the figure of the barracuda is defined as a message as shown in Fig.8. This message contains its lifetime, location and direction of the character, the effective area in which the message is transmitted, and the character who sends this message.

The environment selects messages which a sensor of a character needs and returns those message to the sensor. A character has the following methods:

- (<- self :search-message) for searching the messages from the environment by its sensor(s).
- (<- self :send-message a-message) for sending a message to the environment.

4.4 Programming Environment

The *barracuda and herring school* behavioral simulation program is executed as shown in Fig.9. In this picture, one barracuda and several herrings swim in the sea. There are two windows which show the messages in the current and future pool of the environment. Additionally, there are some buttons named event, sensor, message, environment, etc to monitor the characters, messages and environment. In this case, the sensor button is pressed, and the accessible areas of the sensors are displayed as circles.



```
x = 472 y = 211 z = 0 r = 140 sender = herring status = escape
x = 688 y = 361 z = 0 r = 140 sender = herring status = schooli
x = 509 y = 340 z = 0 r = 140 sender = herring status = schooli
x = 411 y = 476 z = 0 r = 140 sender = herring status = schooli

*** new frame ***
x = 350 y = 226 z = 0 r = 140 sender = barracuda status = attac
          62 z = 0 r = 140 sender = herring status = schoolin
```

```
x = 704 y = 175 z = 0 r = 140 sender = herring status = schooling
x = 455 y = 221 z = 0 r = 140 sender = herring status = escape
x = 707 y = 368 z = 0 r = 140 sender = herring status = schooling
x = 528 y = 347 z = 0 r = 140 sender = herring status = schooling
x = 429 y = 484 z = 0 r = 140 sender = herring status = schooling

*** new frame ***
x = 300 y = 224 z = 0 r = 140 sender = barracuda status = attacking
x = 683 y = 78 z = 0 r = 140 sender = herring status = schooling
x = 485 y = 119 z = 0 r = 140 sender = herring status = escape
x = 685 y = 168 z = 0 r = 140 sender = herring status = schooling
x = 472 y = 211 z = 0 r = 140 sender = herring status = escape
x = 688 y = 361 z = 0 r = 140 sender = herring status = schooling
x = 509 y = 340 z = 0 r = 140 sender = herring status = escape
x = 411 y = 476 z = 0 r = 140 sender = herring status = schooling
```

```
51
8.user>(/ 1024 16
64
9.user>
```

Fig.9 Simulated behavior of a school of herrings attached by a barracuda

5 Discussion

An object in Smalltalk-80 [Goldberg 1983] is an integrated entity composed of a data structure and methods. When an object receives a message, the object invokes a method associated with the message. This object model is considered to be a passive object model, because an object cannot be active unless it receives a message.

Rehearsal [Gould 1984] is a visual programming environment implemented in Smalltalk-80. An educator who is not a programmer can develop educational software using performers on a stage in a metaphor called theater. A performer (object) is a process which is active while it receives a cue (message). Process switching is done in an event-driven order. In Rehearsal, programming is making a script using a mouse and objects which appear on the screen as performers.

Director [Kahn 1978] and ASAS [Reynolds 1982] use the actor model [Hewitt 1977]. An actor is an asynchronous, concurrent entity and is considered to be a passive object model. However, the object model used in Director and ASAS can be called an active object model. Because an actor is invoked in every time unit, while it keeps messages from other actors on the stack in FIFO order. This is different from an actor which cannot receive any messages unless other actors send messages. Let us call this object model a dependent active object model. The purpose of ASAS is computer animation. ASAS has rich 3D graphical functions with which to operate an actor in a 3D world. Everything needed to make a movie such as camera, audio, light, are represented by actors. Programming is considered to be writing an operation sequence in a formal script. ASAS interpreter interprets scripts and sends messages to actors. Each actor behaves under the control of the scripts.

In PARADISE, the characters have behavioral patterns as its knowledge. Such a character can be considered to be another active object model. The character is also controlled in time-slice order. A character object does not receive any messages directly, but voluntarily searches and collects messages. Thus, message passing is indirect, using the environment. Lets us call this object model an independent active object model. The mechanism by which a character object voluntarily searches for messages is similar to that in which a process selects information from a black board in [Hayes-Roth 1979]. In Director and PARADISE, there are no scripts. Programming is representing an object's behavioral pattern as its knowledge.

The most interesting point of PARADISE, compared with the other systems, is that a message in PARADISE is an object, so that it is not related to a message selector. A method to be executed is decided by the matched behavioral rule according to the message objects and current state. The comparison of the object model, script, scheduling, instruction for communication, and the purposes of Rehearsal, ASAS, and PARADISE are shown in Table 1.

Table 1: Comparison of Rehearsal, ASAS, and PARADISE

	kind of object	script	scheduling	instructions for communication	purpose
performer in Rehearsal	passive object	YES	event-driven	direct send	education, visual programming
actor in ASAS	dependent active object	YES	time-slice	direct send, direct send all	computer animation
character in PARADISE	independent active object	NO	time-slice	indirect send, search	behavioral simulation, experiment

6 Conclusion

We have proposed a behavioral simulation model and implemented a behavioral simulation system called PARADISE. Some applications are running and being implemented, such as traffic simulations, a car driver's psychological simulations, and panic evacuation simulations.

The authors would like to express their sincere gratitude to Mr. Michio Isoda of Keio University for his reliable advice. The authors are also grateful to Mr. Hirohisa Ishino and Mr. Junichi Nagahama for implementing the animation display part of the PARADISE system.

References

[Partridge 1982] Partridge, B., *The Structure and Function of Fish Schools,* Scientific American 246(6), 1982.

[Reynolds 1985] Reynolds, C., *Description and Control of Time and Dynamics in Computer Animation,* notes for the SIGGRAPH'85, Symbolics Graphics Division, 1985.

[Tokoro 1984] Tokoro, M. and Ishikawa, Y., *An object-oriented approach to knowledge systems,* Proc of Int'l Conf. on Fifth Generation Computer Systems, ICOT, 1984.

[Ishikawa 1987] Ishikawa, Y., and Tokoro, M., *Orient84/K : An Object-Oriented Concurrent Programming Language for Knowledge Systems,* in Object Oriented Concurrent Programming, A.Yonezawa and M.Tokoro, eds, MIT Press, 1987.

[Maruichi 1987] Maruichi, T. and Tokoro, M., *PANDORA : A Multiparadigm Programming Language/Environment (in Japanese),* Department of Electrical Engineering, Keio University, 1987.

[Yuasa 1985] Yuasa, T. and Hagiya, M., *Kyoto Common Lisp Report,* Research Institute for Mathematical Sciences, Kyoto University, 1985.

[Bobrow 1983] Bobrow, D., and Stefik, M., *The LOOPS Manual,* Xerox Palo Alto Research Center, KB-VLSI-81-13, 1983.

[Forgy 1981] Forgy, C., *OPS5 User's Manual,* CMU-CS-81-135, Carnegie Mellon University, 1981.

[Goldberg 1983] Goldberg, A. and Robson, D., *Smalltalk-80 - The language and its implementation,* Addison-Wesley, 1983.

[Gould 1984] Gould, L. and Finzer, W., *Programming by Rehearsal,* Xerox Palo Alto Research Center, SCL-84-1, 1984.

[Kahn 1978] Kahn, K. and Hewitt, C., *Dynamic Graphics using Quasi Parallelism,* SIGGRAPH'78 Conference Proceedings, published as Computer Graphics 12(3), 1978.

[Reynolds 1982] Reynolds, C., *Computer Animation with Scripts and Actors,* SIGGRAPH'82 Conference Proceedings, published as Computer Graphics 16(3), 1982.

[Hewitt 1977] Hewitt, C., *Viewing Control Structures as Patterns of Passing Messages,* Artificial Intelligence 8, North-Holland, 1977.

[Hayes-Roth 1979] Hayes-Roth, B. and et.al, *Modeling Planning as an incremental, opportunistic process,* IJCAI'79 Conference Proceedings, 1979.