한국시뮬레이션학회 '98추계학술대회 논문집 1998.11.7 청주대학교

자율개체와 이벤트 프로그래밍

조 은 상, 고 형 석

Autonomous Agents and Event Programming

°Eun-Sang Cho, Hyeongseok Ko

Human Animation Center
School of Electrical Engineering, Seoul National University

요 약

One of the eventual goals of VR research is to provide valuable *experiences* to the participants. In this work, we view that the content of experience is composed of a sequence of events, and develop algorithms for authoring those events. Event authoring can be realized by controlling agents in VE in two different modes: (1) the autonomous mode, in which the agent exhibit autonomous behaviors based on the current world status and its own personality, and (2) the event mode, in which the behaviors generated from the autonomous mode is further controlled to meet the needs of the experiment. We define the *event authoring language*, so that the authors can design experiments by writing *event-programs*. Then the architecture of *event execution manager* is described, which is the heart of event-program execution. prove the effectiveness of our approach by showing results of several experiments.

1 Introduction

This paper presents algorithms for generating autonomous agents which are capable of intelligent interactions with the participants and organizing events which can be achieved by controlling the agents. This research was originated from driving simulation. Therefore examples in the following discussion are taken from driving simulation. However, we believe that the result can be applied to other applications as well.

The goal of our driving simulation is not a fast walk-through on the empty roads, but we like to produce interesting events along the way, so that the participants get valuable lessons which can be costly on the actual roads. If the simulation runs long enough, the *ambient cars*[4] will produce all the situations that should be experienced by the *subject car*. In this case, however, we have no direct control over the situation flow. We don't know how long it will take for the subject car to go through all the elements originally planned in the experiment, or some elements may occur repeatedly. On the other hand, we may choose to make the program actively create critical events by controlling the cars around the subject car. We call the flow of events *event-program*, and the process that realizes such flow *event authoring*[7].

In this paper, we propose novel ideas to implement autonomous agents and control those agents to realize desired event-program. We define each agent as a software module. For event authoring, we define a language called *Event Authoring Language* (EAL), and its semantics[1].

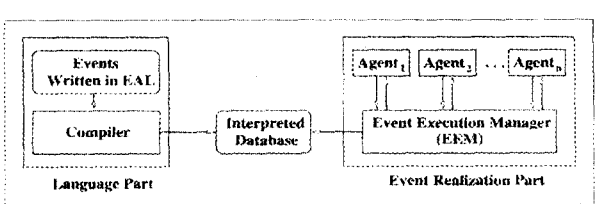

Figure 1: Configuration of the Whole System

2 Related Work

It is not surprising that firm theoretical fundamentals of event authoring have not been

1. We developed a compiler for EAL.

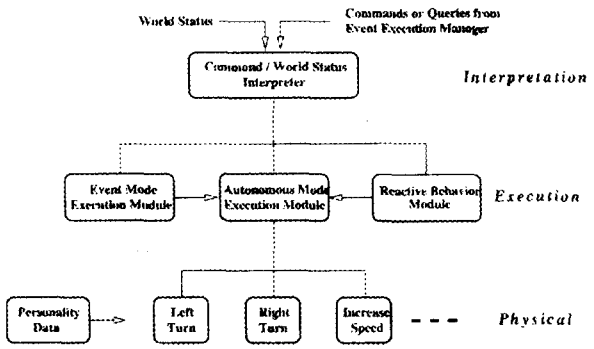

Figure 2: The Architecture of Autonomous Agent

established yet, because VR is a new field of study and the idea of event authoring started recently. However, Perlin and Goldberg[10] have studied the *Improv* whose goal is to give rich interactions to agents. And, Blumberg and Galyean[2] showed impressive results. They made the agent composed of three layers and could achieve the properties of both autonomy and directability. The Iowa Driving Simulator (IDS) project[6][5][3][11][8] in the University of Iowa has much resemblance with our study. The project team made a car as a set of complex states and let the behavior controller concern with the transitions between the states directly. Besides above researchers, others also have done much works on the autonomous agents and the event authoring.[9][1]

3 Architecture

Our event authoring system consists of two parts : the language part and event realization part. In the language part, the EAL compiler parses an event-program written in EAL and produces a database that is directly accessible from the event realization part. The event realization part consists of the agents and event execution manager (EEM).

An EAL file starts by listing the agents involved in the events, and then describes the preconditions and events that constitute the content of the event-program. The event execution manager (EEM) is the heart of the event-program execution. EEM can send commands or information queries to agents. Then agents execute the commands or answers the queries.

4 Modeling Autonomous Agents

The agents should be equipped with basic capabilities, so that the event execution manager can control them by supplying minimum amount of commands. The model should be general, but at the same time it should have a room for diversity.

We subdivide the agent model into three hierarchical layers as shown in Figure 2: *the interpretation, execution, and physical layers*

4.1 The Interpretation Layer

The interpretaion layer recognizes the world status, and interprets incoming commands according to the current context, and sends the information to the execution layer. Whenever a different type of agent is introduced in the simulation, the EEM does not need to be reprogrammed. Instead, the new type of agent should follow the common protocol which is expected by the EEM and pre-existing agent modules.

4.2 The Execution Layer

The execution layer, which functions like the human brain, receives the commands translated by interpretation layer, and determines a sequence of actions that fulfills the given commands, and sends the resulting sequence to the physical layer for animation and visualization. Executive layer is composed of three modules as shown in Figure 2.

- Autonomous Mode Execution Module (AMEM): The execution layer is in the autonomous mode when there is no external command.
- Event Mode Execution Module (EMEM): The execution layer goes into the event mode when the agent's behavior needs to be controlled to run an event-program.
- Reactive Behavior Module (RBM): RBM modifies the action sequence produced from the above two to prevent imminent accidents.

4.3 Physical Layer

The physical layer is placed in the lowest level of agent system. It takes charge of motor level movements, or compute the actual position of the cars considering the vehicle dynamics. The physical layer consists of several modules, each of which implements the different skills of the agent.

5 Event Authoring

In the previous section, we have explained the architecture of autonomous agent. Now we present the main result of our study, the event authoring algorithm. The agent architecture described in Section 4 enables us to develop an event authoring system which is simple, effective, and easy to use.

5.1 Event Execution Manager

The *event execution manager* (EEM) is the heart of event execution program. We design the EEM using a state machine. But, the states are managed in different ways from classical state machines, or Cremer et al.'s HCSM[3]. The EEM creates a state when the state is needed and if the state is no more used, the EEM destroys it. In addition, the EEM is designed so that multiple states are activated at the same time. This feature allows the topology of the state machine to be dynamically changed based on execution result.

The content of a state is composed of predicates and conditions for state transition and the information of predicates and state transition comes from the event authoring language. More details of EEM follows in Section 5.2.

Due to the data abstraction in the interpretation layer of the agent architecture, EEM can be kept simple. The EEM doesn't need to care about the meanings of data. Instead, the EEM sends the signals to agents and leaves the agents to take appropriate actions. This makes the EEM handle variety of agents and event-programs without any modification.

5.2 Event Authoring Language

The event authoring language (EAL) is defined to facilitate event-program creation. The EAL has its own syntax and semantics. Below is a short example. The meaning of this example will be explained in the subsequent sections.

```
Agent car01 {
    model = car;
    max_speed = 1.0;
    carelessness = 0.1;
}
Agent human01 {
    model = human;
}
Event_Program s_01 {
    Precondition {
        achieve(subject, NULL, LOCATE, r1, 4, 0.3);
        achieve(car01, subject, BACKWARD, 10);
        achieve(human01, NULL, LOCATE, -250,0,-10,0,0,-1);
        next_event event_1;
    }
    Event event_1 {
        do(car01, subject, FORWARD, 5);
        do(human01, NULL, STOP);
        next_event {
            next_event_name event_2;
            condition(car01, subject, FORWARD, 20);
            condition(car01, subject, RIGHT, 0);
        }
    }
    Event event_2 {
        do(human01, NULL, GO);
        do(car01, NULL, DECSPEED, 0.02, 0.2);
    }
    Fail {
        condition(human01, subject, COLLISION);
        condition(subject, car01, COLLISION);
    }
    Success {
        duration = 60;
        condition(car01, NULL, LOCATE, r501);
        condition(car01, subject, FORWARD);
    }
}
```

Defining Participating Agents

At first, the agents participating in the events should be listed.

Preconditions

The EEM must first arrange the agents and other entities in the VE into a situation before starting an event-program. For example, to start an event-program in which a car passes the subject car. The car should be first placed at the behind of the subject car.

`Precondition` is used to specify how to arrange the agents. `Precondition` consists of several predicates and branching destination. For example, the predicate `achieve(subject, NULL, LOCATE, r1, 4, 0.3)` in the above example tells that the subject car must be located on the fourth lane of the road `r1`. If all the conditions are satisfied, the EEM goes to `next_event`.(in the above example, `event_1`)

5.2.3 Events

`Events` define the actual content of an event-program which are executed when the preconditions are all satisfied. An Event is defined by a sequence of `do` predicates and `next_event(s)`.

For example, on receiving `do(car01, subject, FORWARD, 5)` as a command, `car01` tries to go in front of the subject car and maintain the distance about 5 meters from the subject car. If `car01` is located in the behind of the subject car, it will increase its speed and pass the subject car.

If all conditions are satisfied, EEM prepares to go to the next event.

5.2.4 Fail and Success

To determine whether an event-program is executed as the author originally intended, `Success` and `Fail` conditions are specified.

6 Experiment

Our event authoring algorithm was implemented on Silicon Graphics Indigo2 R4400 workstation with Maximum Impact graphics. We applied the algorithm to the driving simulation. In this section, we will show two examples produced by our EAL.

6.1 Example 1: Competitive Passing

(Sketched in Figure 3) This example shows the situation in which a car passes the subject car and then, a pedestrian steps into a road. It is reported that drivers tend to increase its speed when they are passed by other cars. These tendency can lead the subject car to hit the pedestrian. The event-program of this has been listed in Section 5.2.

6.2 Example 2: Line Encroachment

(Sketched in Figure 4) On a two lane roads (one lane for each direction), a bus runs very slowly blocking the subject car. In this case, the subject car may be tempted to encroach the center line and pass the bus. But, the EEM sends a number of cars in the other lane to make the pass a tough job.

7 Conclusion and Future Work

In this paper, new architectures for autonomous agents and event authoring system were proposed. To compromise behaviors of the agent in autonomous mode and event mode, we organized the architecture of the agent into three layers: the interpretation, execution, and physical layers. These agents are controlled by receiving predicates from the event execution manager. This approach enables the event execution manager to handle a variety of agents and guarantees the correct execution of event-programs.

Most significant contribution of this paper is that we invented an event authoring language which is simple, easy to use, powerful, and the language is an open solution for constructing event-programs.

In the future, we will expand the event authoring language for describing fuzzy situations.

8 Acknowledgment

This work is supported by Creative Research Initiatives of the Korean Ministry of Science and Technology.

Reference

[1] Ronald C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. pages 105-122, 1990.

[2] Bruce M. Blumberg and Tinsley A. Galyean. Multi-level direction of autonomous creatures for real-time virtual environments. *In the Proceedings of SIGGRAPH 95*, pages 47-54, August 1995.

[3] James Cremer and Stuart Hansen. Hierarchical, concurrent state machines for behavior modeling and scenario control. December 1994.

[4] James Cremer, Joseph Kearney, and Hyeongseok Ko. Simulation and scenario support for virtual environments. *Computer & Graphics*, 20(2):199-206, April 1996. Special Issue on ''Techniques for Virtual Environments''.

[5] James Cremer and Joseph K. Kearney. Scenario authoring for virtual environments. In *Proceedings of IMAGE VII Conference*, Tucson, USA, June 1994.

[6] James Cremer and Yiannis Papelis. The software architecture for scenario control in the iowa driving simulator. In *Proceedings of Computer Generated Forces and Behavioral Representation Conference*, Orlando, USA, May 1994.

[7] James F. Cremer, Joseph K. Kearney, Yiannis Papelis, and Richard Romano. The software architecture for scenario control in the Iowa Driving Simulator. In *Proc. 4th Computer Generated Forces and Behavioral Representation Conference*, Orlando, FL, May 1994.

[8] J. Kuhl, D. Evans, Y. Papelis, R. Romano, and G. Watson. The iowa driving simulator - an immersive research environment. pages 35-41, July 1995.

[9] Pattie Maes. Designing autonomous agents: Theory and practice from biology to engineering and back. pages 1-2, 1990.

[10] Ken Perlin and Athomas Goldberg. Improv: A system for scripting interative actors in virtual worlds. In *Proceedings of SIGGRAPH Conference*, pages 205-216, New Orleans, USA, August 1996.

[11] John A. Biggs Rod Deyo and Pete Doenges. Getting graphics in gear: Graphics and dynamics in driving simulation. In *Proceedings of SIGGRAPH Conference*, pages 317-326, Atlanta, USA, August 1988.

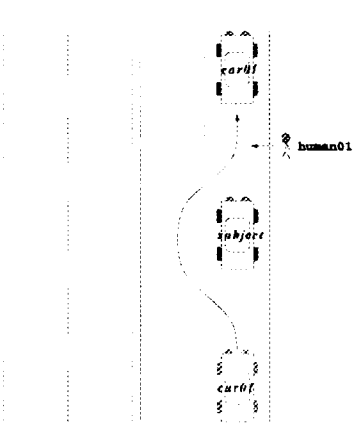

Figure 3: Diagram for Example1

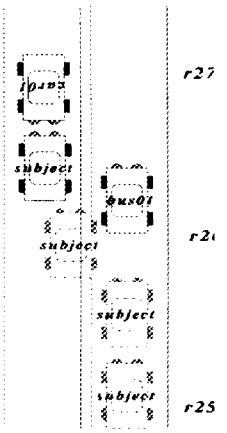

Figure 4: Diagram for Example2