Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

META-LEVEL CONTROL ARCHITECTURE FOR MASSIVELY MULTIAGENT SIMULATIONS

Shohei Yamane

Department of Social Informatics
Kyoto University

Toru Ishida

Department of Social Informatics
Kyoto University

ABSTRACT

Various situations in a massively multi-agent simulation will emerge in a simulation or the period of the simulation will become too long. These situations cause problems for system operators in that each action scenario becomes too complex to maintain and a simulation costs very long time. Therefore, flexible control of the simulation, such as changing simulation speed and switching agents' action scenarios, is required. We propose a meta-scenario description language and a meta-level control architecture. The meta-scenario description language describes how to control simulations and agents based on an extended finite state machine. Meta-level control architecture achieves control on the basis of meta-scenarios provided by a meta-scenario interpreter, which controls interpreters of agents' action scenarios and the simulation environment. In addition, our proposed architecture does not lose scalability of massively multi-agent systems for some applications.

1 INTRODUCTION

The use of high-performance computers and distributed computing environments enable massively multi-agent systems (MMAS) to manage tens of thousands or millions of agents under discussion (Ishida 2005). Traffic simulation is a typical application of MMAS (Balmer 2004). Using MMAS, we can perform city-wide traffic simulations and disaster simulations. Various situations will emerge in such large-scale multi-agent simulation because of the various behaviors of the different agents. In addition, a simulation needs a considerable amount of time because it takes a long time to fully understand the impact of a certain event on the whole simulation. These cases require flexible control of runtime. We mention the following two examples.

First, in large-scale multi-agent simulations, the action scenarios of the agents is very complicated. Generating agent trips is essential for traffic simulations. A method using an OD-matrix (Balmer 2004) and a method using a genetic algorithm (Charypar 2003) are proposed as trip generation methods. Each trip becomes very complicated because various situations can emerge in a large scale simulation. We think it is efficient to simplify each action scenario by preparing a certain action scenario for a certain situation and changing them on the basis of the situation.

Second, to construct more realistic agent models through simulations, it is effective to perform participatory simulations, in which some agents are operated by humans, and construct agent models from the resulting log data (Murakami 2005; Guyot 2005). Participatory simulations are often real-time simulations because the operators are actually making the decisions. However, real-time simulations are ineffective if the simulation period is very long. For this reason, flexible time management, such as making the simulation speed equal real-time only while the operators are making decisions, is required.

To achieve flexible control of large-scale multi-agent simulations, we propose a meta-scenario in which a language is used to describe control of simulations and action scenarios. There are two issues to achieve control based on a meta-scenario.

1. **Definition of control functions**
   We need to define control functions required to describe changing action scenarios and time management.
2. **Development of a control architecture**
   We need a control architecture for simulation control on the basis of meta-scenario descriptions. We will implement this architecture on a MMAS so as not to lose scalability of MMAS.

Section 2 contains the scenario description language that describes the action scenarios of the agents. Section 3 contains an explanation of the meta-scenario. Section 4 contains an explanation of the meta-level control architecture to enable simulation control based on the meta-scenario descriptions. Section 5 contains an application of the meta-scenarios for participatory simulations. Section 6 contains an evaluation of the scalability for the meta-level architecture.

2 SCENARIO DESCRIPTION LANGUAGE

In this paper, we chose scenario description language *Q* (⟨http://www.lab7.kuis.kyoto-u.ac.jp/Q/index_e.html⟩) (Ishida 2002) to describes the action scenarios of the agents. The reason for our selection is outlined in this section.

2.1 Specification

A scenario description describes how an agent should behave and interact with its environment involving humans and other agents. A scenario is represented as an extended finite state machine. The language facilities are explained below.

1. *Cue/Action*. Cues are used to request agents to observe their environment and trigger interaction. Cues are not permitted to have any side effects. Whereas, actions are used to request agents to change their environment. These cues and actions are defined by users, and how the agent that receives a request behaves depends on the implementation of the agent system.

2. *Guarded Command/Scenario*. Guarded commands are used in situations that require the simultaneous observing of multiple cues. A guarded command combines cues and actions. If a cue becomes true, the corresponding action is performed. A scenario is used for describing state transitions, and the state transitions are done by expressions. The scenario defines each state as a guarded command.

```
(defscenario overtake
             (&pattern ($x #f))
 (left-lane-nomal
  ((?carExists :position "ahead"
               :speed "slow" :car $x)
   (!changeLane :lane "right")
   (!overtake :car $x)
   (go right-lane-normal))
  ((?carExists :position "ahead"
               :speed "fast" :car $x)
   (!changeSpeed :speed "fast")
   (go left-lane-fast)))
 (right-lane-normal
  ...)
 (left-lane-fast
  ...)
 ...)
```

   The scenario in the above examples defines the states as left-lane-normal, right-lane-normal, and left-lane-fast. In left-lane-normal, the agent changes its lane and overtakes a car when there is a car driving slowly ahead of the agent, or the agent drives faster when there is a car driving fast ahead of it.

3. *Agent/Avatar*. Agents and avatars are defined by specifying their names. Assigning a scenario to an agent, the agent begins to follow the scenario. Whereas, an avatar is operated by a human. However, we can assign scenarios to an avatar like an agent. By assigning a scenario to an avatar, we can let the avatar behave automatically while the human is not actively making decisions.

2.2 Scenario Processor

A scenario processor interprets a scenario description and requests the corresponding agent to execute cues, actions, and guarded commands. When the cues, actions, and guarded commands are executed, the agent returns the result to the scenario processor. The scenario processor decides the next command on the basis of this result, and then requests the agent to execute the new command. Scenario processors are separated from agent systems. This means that a meta-scenario processor can control the action scenarios of the agents by controlling the scenario processors.

3 META-SCENARIO

The meta-scenario that describes simulation control is explained in this section.

3.1 Definition of Functions

When the operator of an avatar finishes making a decision in a participatory simulation, the decision is not directly reflected in the simulation environment but is reflected in the avatar's internal state in most cases. Therefore, to observe the operator's decision making, the scenario processing state of the avatar must be observed. To change scenarios on the basis of simulation situations, we have to define functions, such as observation of simulation environment, and change scenarios assigned to agents. In addition, we have to define functions that change simulation environment to change simulation speed.

We propose a meta-scenario language with the control functions mentioned above. We used an extended finite state machine to describe the meta-scenario because simulation situations can be represented by states and controls can be represented by state transitions.

Q, as explained in section 2, is also based on extended finite state machine model. Therefore, we used cues, actions, and guarded commands to describe the meta-scenario descriptions as well as the scenario descriptions.

A summary of cues and actions of the meta-scenario description language is listed in Table 1. The commands

Table 1: Cues and Actions of Meta-Scenario Description Language

Scenario processing control	
?observeAction	observe execution of action
?observeCue	observe execution of cue
?observeTransition	observe execution of state transition
?observeScenario	observe execution of scenario
!runCue	request agent to execute cue
!runAction	request agent to execute action
!releaseScenario	release scenario from agent
!assignScenario	assign scenario to agent
Simulation control	
?observeEnvironment	observe simulation environment
!getEnvironment	get environmental information
!setEnvironment	set environmental information
!createAgent	create agent
!deleteAgent	delete agents
!createCrowd	create crowd of agents
!createAvatar	create avatar
!startSimulation	start simulation
!stopSimulation	stop simulation
observe/get information from real-world	
?observeMetaEnvironment	observe information from real-world
!getMetaEnvironment	get information from real-world

in this table that start with ? are cues and that start with ! are actions.

3.2 Examples of Meta-Scenario

The meta-scenario description language can describe various simulation controls. For example, in a traffic simulation, some agents may prefer to take a detour to avoid a traffic jam, whereas others may not. This situation requires two scenarios to be written for each agent. One scenario involves taking the usual route, and the other scenario involves taking a detour. We then write a meta-scenario to change the agents scenario if the agent, who prefers to avoid traffic jams, experiences a traffic jam on their usual route.

For another meta-scenario, we can write a meta-scenario to describe a simulation scenario. That is, the meta-scenario describes events in the simulation, including the creating agents, the deleting agents, the occurrence of traffic jams, and the occurrence of disasters.

Figure 1 shows an example of a meta-scenario when an agent's scenario is changed because of a traffic jam. As shown in Figure 1, each state in the meta-scenario is represented by a circle, and each transition is represented by an arrow. In addition, a cue that triggers a transition and

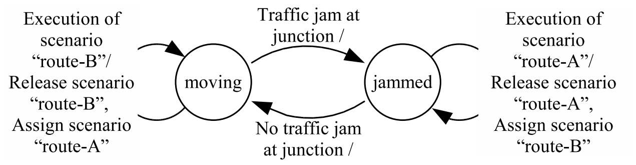
Execution of scenario "route-B"/ Release scenario "route-B", Assign scenario "route-A"
Traffic jam at junction /
moving
jammed
No traffic jam at junction /
Execution of scenario "route-A"/ Release scenario "route-A", Assign scenario "route-B"

Figure 1: State Machine of Meta-Scenario that Changes Agents' Scenarios

an action involved by the transition are indicated in form of (cue/action) – the lies on the appropriate state transition arrow. In this example, the meta-scenario assigns a scenario, "route-A", (take a usual route) when the traffic is moving and assigns a different scenario, "route-B", (take a detour) when a traffic jam occurs.

Figure 2 shows a meta-scenario description of the meta-scenario shown in Figure 1; the cues and actions listed in Table 1 are used.

4 ARCHITECTURE FOR SCENARIO PROCESING CONTROL

The meta-level control architecture to achieve the meta-scenario processor is explained in this section. The control flow for scenario processing is also explained.

Meta-level architecture has been described elsewhere (Gao 2002). Each agent has its own meta-layer to debug scenarios (Gao 2002). In this paper, a meta-level controls an entire simulation, including the environment and agents.

4.1 Meta-Level Control Architecture

Figure 3 shows the meta-level control architecture we designed.

Scenario Description A scenario description outlines an agent's behavior using the language facilities explained in Section 2.

Scenario Interpreter A scenario interpreter corresponds to an agent in an agent system and interprets a scenario description. The result of this processing is a request to the agent to trigger an observation or to perform an action.

Meta-scenario description A meta-scenario describes the control and observation of scenario processing, such as observation and changing the command executed by the scenario interpreter. The meta-scenario also describes the setting of and acquisition of environmental information of the agent system, such as changing simulation speed.

Meta-scenario interpreter A meta-scenario interpreter interprets the meta-scenario descriptions. A meta-scenario requests the scenario interpreter to ob-

```
(defmetascenario avoid-jam
                 (&pattern (agent #f))
  (moving
     ; observe traffic jam
    ((?observeEnvironment
                  :name "traffic jam"
                  :value #t)
     (go jammed))
     ; observe execution of 'route-B',
     ; and set variable 'agent' to
     ; the agent
    ((?observeScenario :name 'route-B
                       :agent agent)
     ; release 'route-B' from agent
     (!releaseScenario :name route-B
                       :agent agent)
     ; assign 'route-A' to agent
     (!assignScenario :name route-A
                      :agent agent)
     (go moving))
  (jammed
     ; observe the traffic is moving
    ((?observeEnvironment
                  :name "traffic jam"
                  :value #f)
     (go moving))
     ; observe execution of 'route-A'
     ; and set variable 'agent' to
     ; the agent
    ((?observeScenario :name 'route-A
                       :agent agent)
     ; release 'route-A' from agent
     (!releaseScenario :name 'route-A
                       :agent agent)
     ; assign 'route-B' to agent
     (!assignScenario :name 'route-B
                      :agent agent)
     (go jammed)))
```

Figure 2: Meta-Scenario Description that Changes Agents Scenarios

serve the scenario description processing, requests an agent to trigger an observation or perform an action, and requests the agent system to set/get environmental information.

Agent System An agent system receives and processes all the requests for the agents.

One meta-scenario interpreter in our meta-level control architecture controls all scenario interpreters. As an alternative architecture, we can consider that an individual meta-scenario interpreter controls an individual scenario interpreter. This alternative architecture has an advantage in controlling scenario interpreters because each scenario interpreter has its own meta-scenario interpreter. However, meta-scenario interpreters in such an architecture need to coordinate to control the simulation environment. For example, when a meta-scenario interpreter is performing the simulation at speed slow while another is it performing fast, they need to communicate to determine what the writer really wants. The cost of coordination will be high, and the result may be wrong.

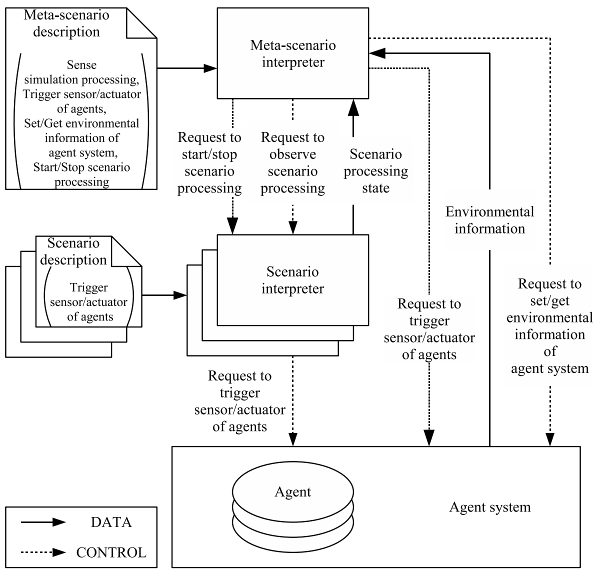
Meta-scenario description
Sense simulation processing, Trigger sensor/actuator of agents, Set/Get environmental information of agent system, Start/Stop scenario processing
Meta-scenario interpreter
Request to start/stop scenario processing
Request to observe scenario processing
Scenario processing state
Environmental information
Scenario description
Trigger sensor/actuator of agents
Scenario interpreter
Request to trigger sensor/actuator of agents
Request to set/get environmental information of agent system
Request to trigger sensor/actuator of agents
Agent
Agent system
DATA
CONTROL

Figure 3: Meta-Level Control Architecture

We can assign more than one meta-scenario to one simulation. For example, using a meta-scenario to change the simulation speed and a meta-scenario to change scenarios enables time-management and scenario switching. Unfortunately, coordination becomes a problem when multiple meta-scenario interpreters are run.

4.2 Flow of Meta-Scenario Execution

We will explain the flow in meta-scenario processing for changing the actions of a scenario. The changing of actions is described using `?observeAction` and `!runAction` commands. We described an example that changes the scenario when a traffic jam occurs in section 3.2. In this subsection, we assume that one action can describe a route an agent take (e.g., `(!takeRoute :route route-A)` is an action to request the agents to take route-A). Figure 4 shows a sequence diagram in which the meta-scenario change the action from a usual route to a detour.

1. The meta-scenario interpreter reads the meta-scenario description and determines the next command to be executed. The meta-scenario interpreter in this example requests all scenario interpreters to observe the action taken for the usual route.
2. The scenario interpreter reads its scenario description and decides the next command to be executed.
3. The scenario interpreter tests whether the command determined in step 2. is observed by the meta-scenario interpreter (i.e. the action taken is the usual route).

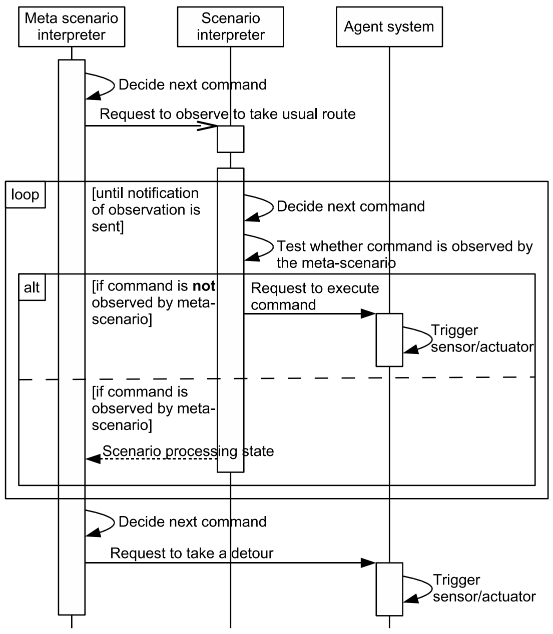
Meta scenario interpreter
Scenario interpreter
Agent system
Decide next command
Request to observe to take usual route
loop
[until notification of observation is sent]
Decide next command
Test whether command is observed by the meta-scenario
alt
[if command is **not** observed by meta-scenario]
Request to execute command
Trigger sensor/actuator
[if command is observed by meta-scenario]
Scenario processing state
Decide next command
Request to take a detour
Trigger sensor/actuator

Figure 4: Sequence Diagram of Scenario Execution Control

4. If the command is not observed, the scenario interpreter requests the agent to execute sensor/actuator and returns to step 2.
5. If the command is observed, the scenario interpreter sends a scenario processing state to the meta-scenario interpreter.
6. The meta-scenario interpreter receives the observation result, reads the meta-scenario description, and determines the next command to execute.
7. In the example shown in Figure 4, the next command is to request the agent to take the usual route to execute an action that takes a detour.

The costs associated with implementing meta-level control architecture are expensive, therefore we have to reduce the extra cost of meta-level control. The extra cost is mainly caused by observation/control of scenario interpreters. If a meta-scenario observes and controls all the processes of each scenario interpreter, we can achieve more flexible control. However, the overhead with implementing such a system becomes quite high because message transfer between the meta-scenario interpreter and the scenario interpreter occurs at each step of the scenario interpreters. Each scenario interpreter in the proposed implementation contains its own scenario controller. The meta-scenario requests the scenario controllers to observe and control the scenario processing, the scenario controller observes and controls the scenario interpreter, and the scenario interpreter then sends the results to the meta-scenario interpreter. In this manner, message transfer between the meta-scenario interpreter and the scenario interpreter occurs only when observation or control is done.

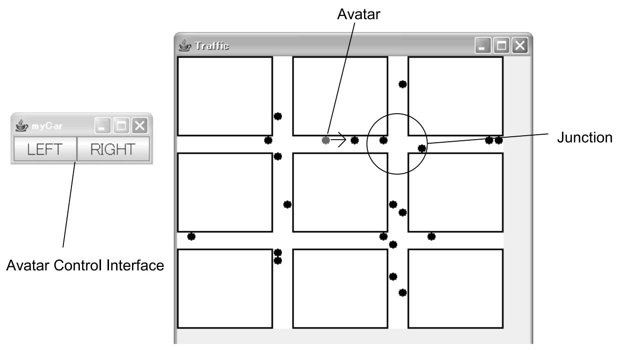

Figure 5: GUI of Simulation

5 APPLICATION TO A PARTICIPATORY SIMULATION

An example of using a meta-scenario in a participatory simulation is described in this section.

Some agents, called avatars, are operated by humans in participatory simulations. The simulation speed of a participatory simulation must be slowed (or raised) to real-time speed while the operators are making decisions. With the meta-scenario, time-management like this is achieved by controlling the simulations on the basis of observations of the avatars and simulations.

5.1 The Simulation Performed in This Section

A graphical user interface of the simulation is shown in Figure 5. The right window shows the road grid. The left window shows the user interface for the operator of the avatar. In this simulation, many agents are driving on a road grid, with one avatar driving straight along a road. The human operating the avatar is assumed to make a decision when the avatar approaches the next junction (indicated by the circle shown in Figure 5). The operator makes his decision and clicks the “LEFT” or “RIGHT” button as desired.

5.2 A Scenario for an Avatar

The state transition diagram of the avatar scenario is shown in Figure 6. In the absence of any direct order from the operator, the avatar keeps driving straight if there is no car in front of it; if there is such a car, the avatar stops.

As shown in Figure 6, the scenario switches to the `turn-left` state when the operator clicks the “LEFT” button, alternatively the scenario switches to the `turn-right` state when the operator clicks the “RIGHT” button. Regardless of when the `turn-left`/`turn-right` state is

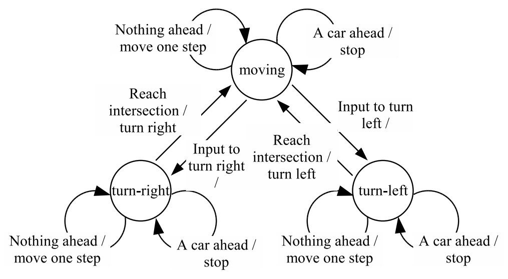
Nothing ahead / move one step
A car ahead / stop
moving
Reach intersection / turn right
Input to turn left /
Input to turn right /
Reach intersection / turn left
turn-right
turn-left
Nothing ahead / move one step
A car ahead / stop
Nothing ahead / move one step
A car ahead / stop

Figure 6: Scenario for Avatars

entered, the avatar keeps going straight until it reaches the intersection where it can make the selected turn. Once the turn is completed, the scenario transits to the moving state.

Assigning a scenario in this manner to an avatar can automate its behavior. This frees the operator from having to make all decisions, like stopping the avatar to avoid a collision.

5.3 A Meta-Scenario of Time-Management

In this simulation, we assumed that the operator of the avatar needs to make a decision only when the avatar approaches a junction. Figure 7 shows a meta-scenario when the simulation speed is slowed as the avatar approaches a particular junction and when the simulation speed is increased once the operator has finished making his decision (by clicking one of the buttons). The state transition diagram of this meta-scenario is shown in Figure 8.

This meta-scenario runs in the following way. First, the simulation is initialized in the `initial` state and transits to the `on-road` state. In the `on-road` state, when the avatar is far from the junction, the simulation speed is faster than real-time. When the avatar approaches the junction, the simulation speed is slowed, and the meta-scenario state transits to the `near-junction` state. If the operator has made a decision, the state of the avatar scenario transits to the appropriate state. The operator's decision is detected by using a `?observeTransition` command in the `near-junction` state. Once the decision is detected, the simulation speed is increased. To prevent the `near-junction` state from being unnecessarily reentered (which would trigger a slow down in simulation speed), the meta-scenario waits some time after the avatar leaves the junction in the `within-junction` state.

As shown in this example, the operators' decision making is not always directly reflected in the simulation environment. However, the decision should be reflected in the avatars' internal states. The meta-scenario control architecture can achieve time management by utilizing scenario processing, such as state transitions of scenarios. This example shows that monitoring and controlling the scenario processing of agents at the meta-level enables simple and flexible control of agents and simulations based on observations from simulation processing, including the agents' behavior.

```
(defmetascenario meta-traffic ()
  (initial
   (#t
    (!createCrowd :name 'Cars :population 20)
    (!createAvatar :name 'myCar)
    (!assignScenario :name 'car :agent Cars)
    (!assignScenario :name 'car-avatar
                     :agent myCar)
    (!startSimulation) (go on-road)))
  (on-road
    ; observe myCar to approach the junction(2,1)
   ((?observeEnvironment :name 'onIntersection
                         :args (list myCar 2 1))
    ; slow simulation speed
    (!setEnvironment :name 'sim-speed
                     :args "slow")
    (go near-junction)))
  (near-junction
    ; observe myCar to leave the junction(2,1)
   ((?observeEnvironment :name 'outOfIntersection
                         :args (list myCar 2 1))
    ; raise simulation speed
    (!setEnvironment :name 'sim-speed
                     :args "fast")
    (go on-road))
    ; observe myCar to transit to 'turn-right'
    ; in the scenario for the avatar 'car-avatar'
   ((?observeTransition :scenario 'car-avatar
                        :scene 'turn-right
                        :agent myCar)
    ; raise simulation speed
    (!setEnvironment :name 'sim-speed
                     :args "fast")
    (go within-junction))
    ; observe myCar to transit to 'turn-left'
    ; in the scenario for the avatar 'car-avatar'
   ((?observeTransition :scenario 'car-avatar
                        :scene 'turn-left
                         :agent myCar)
    ; raise simulation speed
    (!setEnvironment :name 'sim-speed
                     :args "fast")
    (go within-junction)))
  (within-junction
    ; observe myCar to leave the junction(2,1)
   ((?observeEnvironment :name 'outOfIntersection
                         :args (list myCar 2 1))
    (go on-road))))
```

Figure 7: Meta-Scenario Description of Time-Management

6 EVALUATION

We showed how to control simulations by using the meta-scenario in the previous sections. The aim of a meta-scenario is to enable flexible control of large-scale simulations. As stated in Section 4, scalability is also an important issue. Unfortunately, the system used in Section 5 is a prototype, and we have not developed a system with enough scalability. We have developed a system that can manage some hundreds of thousands of scenario interpreters using Caribbean (Yamamoto 2001), which was developed by IBM. Therefore, we can create a MMAS with a meta-scenario if the cost of meta-level control is low.

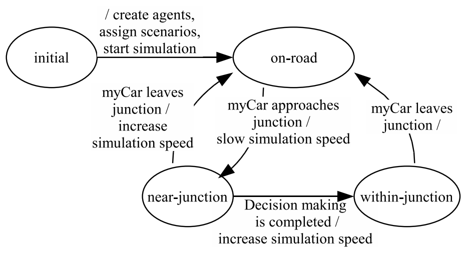
/ create agents,
assign scenarios,
start simulation
initial
on-road
myCar leaves junction / increase simulation speed
myCar approaches junction / slow simulation speed
myCar leaves junction /
near-junction
within-junction
Decision making is completed / increase simulation speed

Figure 8: State Machine of Meta-Scenario for Time-Management

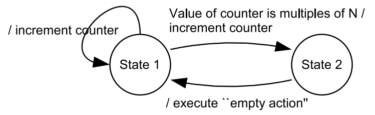
/ increment counter
Value of counter is multiples of N / increment counter
State 1
State 2
/ execute ``empty action"

Figure 9: Scenario for Evaluation

6.1 Experiment

As explained in Section 4, the extra cost of meta-level control is mainly caused by observation/control of scenario interpreters. We created the following test to evaluate the overheads.

- The number of agents is 100.
- Each agent had a counter in its own memory.
- An agent scenario used is shown in Figure 9.
- The simulation ran for in one minute, and the sum of the counters of all the agents was measured.
- We performed two simulations. One was controlled by the meta-scenario and another was not. Then we compared them. The meta-scenario observed an "empty action", as shown in Figure 9.

The scenario in this test was very simple. The scenarios for real applications will be more complex, and the cost of executing each cue and action will be higher. However, if a (meta-)scenario is very complex, the calculation cost of (meta-)scenario processing will not increase very much. This is because the fundamental process of a (meta-)scenario interpreter is not related to the size of the (meta-)scenario. Whereas, the cost to execute each cue or action differs among applications. The cost of the cues and actions in the test we performed was very small. This test was almost the worst case. In real applications, the ratio of the meta-level control cost to the overall cost would be smaller.

6.2 Result

Results are listed in Table 2. The number in each cell in this table means the sum of the overall counters, and the parenthetic values are the number of observations by a meta-scenario per second. The results listed in Table 2 show that when the observation frequency is low, (in the case N = 10 and N =100) a little decrease in performance is observed. This is because no message was sent while no observations were made a by a meta-scenario (execution of "empty action") during our implementation. When the frequency is high (N = 1), some overhead was observed (performance becomes about 2/3) because the meta-scenario had to process the observation results.

Table 2: Evaluation Results

Rate of observed action	Total number of actions with meta-scenario	Total number of actions without meta-scenario
1	20238(337.30)	30361
0.1	34277(57.13)	34708
0.01	33934(5.66)	35183

These results suggest that if the number of target observations by a meta-scenario is small, the influence of the meta-scenario on performance is low. For example, only avatars are observed in participatory simulations. When applications change scenarios, many messages are sent at the time of changing. Other than at this moment, however, most of the simulation is performed with little overhead. As a result, this overhead has little influence on the simulation performance as a whole. Therefore, the architecture shown in Figure 3 is appropriate for the applications, such as participatory simulations and traffic simulations, we considered.

7 CONCLUSION

To perform traffic simulations and participatory simulations in large scale, flexible runtime control of simulations is required. We accomplished the following two goals.

1. To develop a meta-scenario description language
   We defined functions required to describe observe and control simulations and scenario processing.
2. To develop meta-level control architecture
   A meta-scenario interpreter in our proposed architecture controls the scenario interpreters that are separate from the agents system.

Using examples, we showed a meta-scenario can describe time management and scenario changing. We also showed this architecture provides scalability in applications, such as participatory simulations and traffic simulations.

REFERENCES

Balmer, M., N. Cetin, K. Nagel, and B. Raney. 2004. Towards truly agent-based traffic and mobility simulations. International Joint Conference on Autonomous Agents and Multiagent Systems. 60–67.

Balmer, M., and M. Rieser. 2004. Generating daily activity chains from origin-destination matrices. Transportation Research Board 84th Annual Meeting.

Charypar, D., and K. Nagel. 2003. Generating complete all-day activity plans with genetic algorithms. presented at the 10th International Conference on Travel Behaviour Research.

Gao, Z., T. Kawasoe, A. Yamamoto, and T. Ishida. 2002. Meta-level architecture for executing multi-agent scenarios. Pacific Rim International Workshop on Multi-Agents. 163–177.

Guyot, P., A. Drogoul, and C. Lemaitre. 2005. Using emergence in participatory simulations to design multi-agent systems. Proceedings of The Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems. 199–203.

Ishida, T. 2002. Q: A scenario description language for interactive agents. IEEE Computer **35** 42–47.

Ishida, T., L. Gasser, and H. Nakashima, eds. 2005. Massively Multi-Agent Systems I. Lecture Notes in Artificial Intelligence, 3446. Springer-Verlag.

Murakami, Y., Y. Sugimoto, and T. Ishida. 2005 Modeling human behavior for virtual training systems. The Twentieth National Conference on Artificial Intelligence. 127–132.

Yamamoto, G., and H. Tai. 2001. Performance evaluation of an agent server capable of hosting large numbers of agents. In: International Conference on Autonomous Agents. 363–369.

AUTHOR BIOGRAPHIES

SHOHEI YAMANE is a master student of the Department of Social Informatics in Kyoto University. In 2005 he graduated from Graduate school of Engineering in Kyoto University. His e-mail address is ⟨yamane@ai.soc.i.kyoto-u.ac.jp⟩, and his web page is ⟨www.ai.soc.i.kyoto-u.ac.jp/~yamane/index_e.html⟩.

TORU ISHIDA is a professor of Kyoto University, IEEE fellow, and a guest professor of Shanghai Jiao Tong University. He has been working on autonomous agents and multiagent systems for more than fifteen years. He is currently working on massively multi-agent systems and semantic Web services He initiated the Digital City Kyoto, the intercultural collaboration experiments (ICE) with Chinese, Korean, Malaysian colleagues, and the Language Grid project. He is an associate editor of Kluwer Journal on Autonomous Agents and Multi-Agent Systems and Elsevier Journal on Web Semantics. His e-mail address is ⟨ishida@i.kyoto-u.ac.jp⟩, and his web page is ⟨www.ai.soc.i.kyoto-u.ac.jp/~ishida/⟩.