Application of Parallel Scenario Description for RoboCupRescue Civilian Agent

Kousuke Shinoda[1,2], Itsuki Noda[1,2,3], Masayuki Ohta[1], and Susumu Kunifuji[2]

[1] Cyber Assist Research Center, AIST, 135-0064 Aomi, Koto-ku Tokyo, Japan
[2] Japan Advanced Institute of Science and Technology
923-1292 asahidai, Tatunokuti Nomi Ishikawa, Japan
[3] PRESTO, Japan Science and Technology Corporation

Abstract. We propose a novel agent framework to describe behaviors of the general public in rescue simulations and implement an application for "Risk-Communication for disaster rescur". Conventional agent description languages are designed to model intellectual behaviors of human that solve a task to achieve a single goal. In a disaster situation, however, it is difficult to model civilians' behaviors such as goal-oriented problem-solving. Instead of such a formalization, we introduce the "Parallel Scenario Description" approach that models agents' behavior as an action pattern or plan of situations. We call these "Scenarios". In the proposed framework, behaviors are divided into multiple scenarios for each goal by *Posit* and Posit operator, in which behavior rules are grouped based on situations where the rules are active. The problem solver PS^2 constructs a rule-set of behavior dynamically according to the situation of the environment and the agent's state. The framework is implemented as civilian agents for RoboCupRescue Simulation to adapt to a general civilian simulation. Moreover, we implemented refuge simulation for disaster rescue simulations to realize "Risk-Communication".

1 Introduction

RoboCupRescue Simulator[1, 2] is designed to maximize contribution to the society and attain high throughput in research. We aim to utilize this simulator for "Risk-Communication" of a disaster rescue. That is, it is very important to simulate civilian agents suitably for simulating realistic phenomena in disaster simulation. This is true because furthers that civilian agents comprise the majority of agents in the simulated world and behave for multiple and ambiguous purposes. Moreover, the general public are themselves, our target people.

In this paper, we model the general public as agent suitably for disaster rescue simulator (RoboCupRescue Simulator) and implement some applications. In disaster rescue simulation, civilian are numerous elements: calculation cost becomes an obstacle to scalability. Moreover, human actions are very complicated. One important attribute of autonomous agents is the ability to behave for multiple and ambiguous goals in rescue simulation, including disaster rescue simulation. Therefore, the behavior design of civilian agents is a key issue from the viewpoint of disaster rescue simulation.

D. Polani et al. (Eds.): RoboCup 2003, LNAI 3020, pp. 628–636, 2004.
© Springer-Verlag Berlin Heidelberg 2004

The "Risk-Communication" is promotion for education about some risk related with various phenomena. We aim to perform "Risk-Communication" using RoboCupRescue simulator in that disaster rescue is important. It is important for applications about disaster rescue to design many civilian agents. So, we implement a civilian agent for RoboCupRescue Simulator and simulate various situations used by these agents.

2 Behavior Description: Parallel Scenario Description

We motivate "Parallel Scenario Description" by describing the difficulty of agent design of a civilian agent in RoboCupRescue Simulation. We introduce an agent framework, that we call "Parallel Scenario Description". This framework has the following features: 'Scenario-based Behavior Description', 'Light-weight Processing', and 'Prototyping Programming'.

Agents' behaviors have various styles from reflective action to deep-thinking decision making accompanied with long-term plans. The agent behavior descriptions are expressed as reactive planning [3] or deliberative planning [4, 5]. However, these planning approaches present some problems. The former has difficult describing long-term planning; the latter presents difficulty in maintaining true expression of an environment and wastes large amount of calculation time in identifying each condition of action rules.

In rescue simulation, on the other hand, human tend to behave as a custom action patterns and plans of a heuristic knowledge to achieve a specific task. It is not easy to decide own action with deep-thinking. In this research, we model human behavior as a collection of such action patterns or plans, and call it a "scenario"; We then behavior description by **scenario-based behavior description**. Scenario-based behavior description is based on scenarios. A "scenario" is an action pattern or plan to achieve a specific task, e.g, "move to a safe location" and "look for refuge". With individual scenarios, designers of agent behavior(scenario-writer) write action rules that are utilized for achieving a specific goal.

We propose *Posit* and PS^2 for an agent framework based on "Parallel Scenarios Description". *Posit* is rule description syntax and PS^2 is problem solver, which accomplishes multiple purposes(scenarios) in parallel, suitable for simulation of civilian. Civilian agent is implemented with *Posit*, Posit Operator and PS^2.

2.1 Posit: Part Of SITuation

Posit is a framework of behavior rule description for PS^2, and a behavior rule set under a specific situation. *Posit* means a piece of situation of agent. "Scenario" is a set of *Posit* and edges, which represent a temporal transition between each *Posit*. Figure 1 shows syntax of *Posit*.

Action rules, in this *Posit*, consist of *Condition*, *Activation* and *Action*. These rules are applicable in the specific situation, and are candidates for firing when

```
Posit ::= (defposit PositName Rule* )
Rule ::= (defrule RuleName  :condition CondiForm
                            :activition   ActivForm
                            :action     ActioForm )

CondiForm ::= ([LogicOP] CondiForm*)|(InSensor ...)
ActivForm ::= CalcForm
ActioForm ::= ([LogicOP] ActioForm*)|(OutSensor ...)
LogicOP ::= and | or | not | progn
InSensor ::= ⟨input sensor name with prefix "?"⟩
OutSensor ::= ⟨output sensor name with prefix "!"⟩
CalcForm ::= ⟨number⟩|Var|(CalcOP CalcForm CalcForm)
CalcOP ::=  + | − | ∗ | / |%
Var ::= ⟨variable name with prefix "*"⟩
```

Fig. 1. Syntax of Posit

the *Condition* is satisfied. *Activation* shows a value function, which is expressed as a numerical value or formula of *Action*. This *Activation* value is utilized as a value for conflict resolution in decision-making. Finally, the agent process system executes the *Action*'s rule that is selected by some selection rule[1]. The *Action* part has two types of operations: one is primitive action, e.g., move, search, say, and so on; the other is *Posit* Control operator. The primitive action is used to request an agent to output to external environment mainly. The latter will be able to manage an internal environment. In *Posit*,a word starting with "?" indicates a passive action; the same one with "!" is a positive action, and "*" is variable. These are shown Fig. 1.

2.2 Posit Operator

Posit is a collection of rules, which are usable in a spacific situation. This means that *Posit* is a piece of situation, which agent faces. That is, the current situation of an agent is expressed as a set of *Posit* is active. In this study, we call this set the "current-situation". Posit operator the manages "current-situation" by replacing *Posit* in accordance with the situation and means temporal-transition of the situation of an agent. Scenario-writer writes these operations in *Posit* clearly. An Agent can control its own rule candidates used by this *Posit* operator.

[1] e.g., max selection, roulette selection

Posit controller has two basic operators: one is (add_posit ...), the other is (remove_posit ...). In addition, there is reserved operator; e.g. (transit_posit A B), (start_scenario S), or some other.

2.3 Parallel Scenario and PS^2: Parallel Scenario Problem Solver

An agent's simple behavior with *Posit* can be shown using a state-transition diagram. This state-transition diagram shows the basic routine-work or behavior pattern which accomplishes a single goal or a simple task in the short term. We call its state-transition a "Scenario". An agent behavior represents a collection of *Posit* and transitions among them by Posit operator. It is changed by changing the component of its collection.

PS^2 is an inference engine that infers the manipulate *Posit*s and determine behaviors according to Scenarios. The key concept of PS^2 is *situated-ness* evaluation, that is, not to say that rules are evaluated in a cycle; only rules in active Scenarios are evaluated, where activeness of *Posit*s are handled explicitly as situational manipulations. In addition, activation of a *Posit* is operated by another *Posit* action slot specifically.

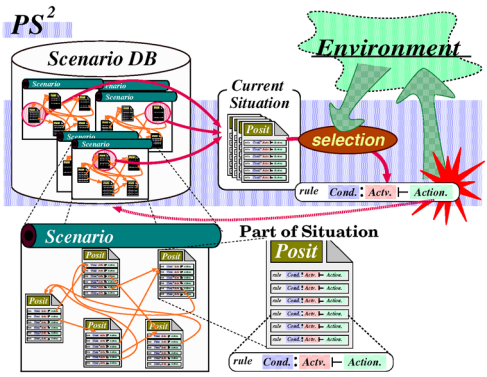

Fig. 2. The processing flow of Decision Making of an Agent's Behavior with PS^2

In PS^2, an entire scenario is stored in "Scenario DB", a form by which a set of active *Posit* is selected into "Current Situation"(CS). Only rules in CS are candidates to be applied in the current situation. Figure 2 shows a flow of information and control in PS^2. A basic cycle to determine one's own action is as follows: Condition parts of all rules in CS are tested for the current environment. Then, for each rule that passes this test, the activation value of these rules is calculated under the current situation. Finally, a rule is selected used by a value of *Activation*, and fire. This basic cycle is repeated until no rule can be applied in the current environment.

3 Application for Social Simulation

3.1 Civilian Agent for RoboCupRescue

Here, we would like to providing explanation of application with our proposed agent architecture: Civilian Agent for RoboCupRescue. Figure 3 shows construction of modules for civilian agent.

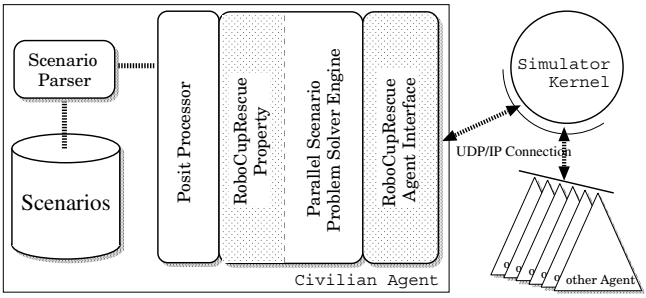

Fig. 3. CivilianAgent Architecture of RoboCupRescue Simulator

The agent consists of several modules. First, the basic level module is "RoboCupRescue Agent Interface", which is provided by RoboCupRescue Simulator as a sample program. It has library modules for connecting with a simulator kernel and a parser of communications protocol, etc. Next, level one is a PS^2. This module needs a specific property definition for RoboCupRescue Simulator to implement this. Other modules are: *Posit* Processor, which manages a current-situation, and which interprets scenarios.

We extended the definition of *Posit*, is mentioned by Fig. 1, to RoboCupRescue Simulator implementing of civilian agent. A module with the necessity of implementing for civilian agents newly, in the module which constitutes an agent is the area, which is covered by gray color, of Fig. 3. We show syntax of *Posit* is the extended definition for RoboCupRescue Simulator, as follows:

3.2 Refuge Action Simulation

We implement a sample application to simulate a refuge action. It is an agent who takes refuge during a disaster. This sample aims at simulating a refuge the general public when an urban disaster is assumed to have occurred; it utilizes RoboCupRescue Simulator as a field of simulation. Concretely, we assume implementation of three kinds of refuge methods: a safe refuge, a guided point refuge, and an instruction absorption refuge. We intend to find from minimum experimentation that refuge time was different by each refuge method. An example of civilian behavior and its *Posit* is as follows:

```
InSensor ::= ?see | ?hear | ?know
OutSensor ::= !search | !move | !say
LocalData ::= (Object :cond CompForm)
Object ::= building :x :y :id :dist :fieryness :broken
         | refuge :id :dist
         | road :x :y :id :dist :width :brocked
         | humanoid :x :y :id :dist :stamina :hp :damage :buriedness
         | world :time
         | self :x :y :id :position :stamina :hp :damage :buriedness
CompForm ::= (CompOP CalcForm CalcForm)
CompOP ::= < | <= | == | >= | >
```

Fig. 4. Definition of Local Data for RoboCupRescue

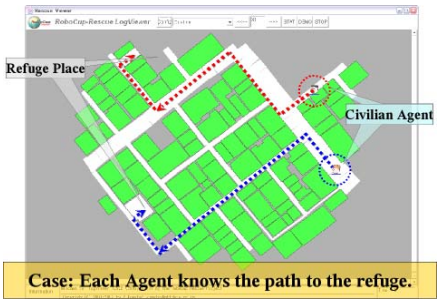

Fig. 5. 'Agent A' and 'Agent B': both civilian knew the surrounding map

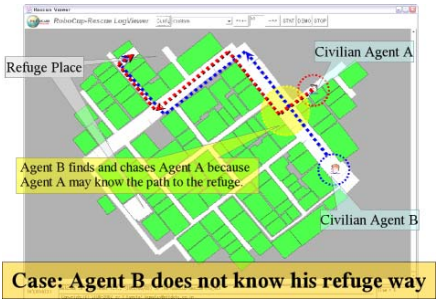

Fig. 6. Agent B: this agent did not know surrounding map, but 'Agent A' knew surrounding map detail, so that 'Agent B' chases 'Agent A' to find the refuge

```
(defposit search_chase_target
  (defrule select_chage_target
    :condition (?see ‘‘people A’’)
    :activity 10    :action (!move to: ‘‘position of its people’’))
  (defrule s_chase_target
    :condition (and (?know ‘‘people A’’)(?see ‘‘his position’’))
    :activity 20    :action (!move to: ‘‘his position’’))
  (defrule clear_chase_target
    :condition (and (?know ‘‘people A’’)(not(?see ‘‘his position’’)))
    :activity 20    :action ‘‘lost a target’’))
```

Figure 5 & 6 shows the refuge action of an agent. Their difference is their own detailed knowledge of the surrounding map.

4 Discussion

4.1 Contribution of Parallel Scenario Description for a Disaster Rescue Simulation Agent

This paper presents a novel agent architecture for large-scale agent-based simulation. Key contributions include: (i) Situated-ness description, (ii) Scenario-based description, (iii) Parallelism and Multiplicity of ordinary human behavior.

These contributions provide several merits for large-scale agent-based simulation. First, a scenario writer is able to design agent behavior in each situation independently according to the scenario and to define dependence among scenarios later. Second, scenarios can be defined individually, and join agent behavior later. Thus, the scenario writer can complicate agent behavior by degrees as a prototyping programming. Third, our framework selects appropriate rules according to circumstances, so that this provides light-weight processing for agents' processes. Fourth, *Posit* is a set of rules under a specific situations. Scenario-writer reuse a *Posit* in other scenarios.

4.2 Related Works

Among related works, our proposed framework differs from conventional system: production systems, Soar and ACT-R, in that these systems aim to model intellectual behavior of humans that solve tasks to achieve a simple goal effectively. In contrast, our approach aims to express knowledge of human behavior as a scenario.

A reactive system: RAPs[3, 6], may enable application to represent parallel scenarios. In RAPs, a task is deconstructed into several sub-tasks that can be executed in parallel. However, RAPs does not offer flexible mechanisms to resolve conflicts of application of multiple rules like activation in *Posit*. For example, it is hard to implement behaviors for an emergency situation that override all other tasks and behaviors. Compared with this, we can realize such behaviors by adding comprehensive activation values to rules for the emergency.

The scenario-type description of agents' behavior in *Posit* was inspired from the work of Interaction Design Language Q[7, 8]. Language Q can handle multiple scenarios. However, its behavior description is complicated because Q has only a "goto" operator. Figure 7 shows, for example, the difference of creating a behavior combining scenario A and scenario B. In the case of *Posit*, a scenario-writer need only add a new edge between A and B. In the case of Q, however, a scenario-writer sets new scenario is combined A and B, and creates complex state-transition. Thus, when scenario-writer wants to add new scenario, writer needs to consider an existent scenario and add new condition in one. Compared with this, scenario writer can create new additional scenario independently, and add several state-transition edges among existent scenarios and new ones in our framework.

Posit shares the mechanism to handle "situated-ness" with GAEA[9, 10]. Both works treat a complex situation as a collection of atomic situations pieces;

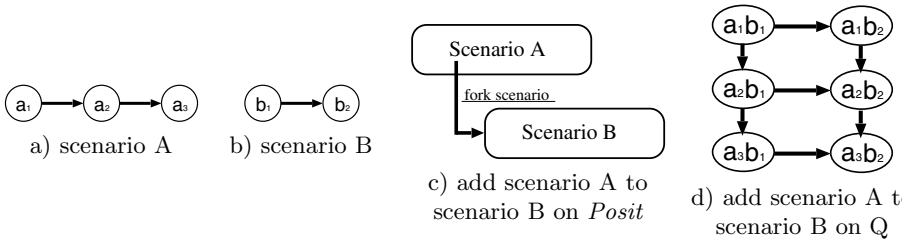

Fig. 7. The difference of rule description flexibility between Q and *Posit*: Agent behavior becomes complex

final behaviors then emerge as composites of atomic behaviors from these situations. However, because GAEA includes full implementation of Prolog, it is too heavy to apply to numerous agents' simulation.

5 Conclusion

This paper proposed a novel agent behavior description and framework using "Parallel Scenarios Description". Agent description languages have been studied intensively, but are not yet proposed as a convenient language used for RoboCupRescue simulation. To utilize knowledge of social scientists, we developed *Posit* (Part Of SITuation) and PS^2 (Parallel Scenarios Problem Solver), a parallel scenario description language, to capture behavior patterns in a specific situation, and to connect edge each small scenario.

We implemented some disaster rescue agent using its behavior model, because our proposed aims to model general public, who has multiple and ambiguous goals, as civilian agent. Concretely, we implemented a civilian agent for RoboCupRescue and a disaster refuge simulation as an application. We intended to use civilian agnet as agent simualtion tools for Risk-Communication, etc.

Moreover, our approach has some demerits. It is not easy to write behavior rules on a text-based interface. Moreover, it is difficult to decide activation values for each rule considering the relation among different *Posit*, and so on. That is, we focus on several point as Feature Work. as following:

- Communicate model base on RoboCupRescue Civilian ACL[11]: To do cooperation work with other rescue agent.
- Another application: for example, Rescue Agent etc.
- Development Environment:To promote of civilian agent development for social scientist, Visual Programming Environment etc..
- Learning Mechanism: To change *Activation* function to acquire appropriate action of civilian agent, and management balance of each scenario.

References

1. Kitano, H.: Robocup rescue : A grand challenge for multi-agent system. In: Proceedings of the Third International Conference on Multi-Agent Systems(ICMAS-2000). (2000) 5–11
2. Rescue HP: (`http://www.r.cs.kobe-u.ac.jp/robocup-rescue/`)
3. Firby, R.J.: Adaptive Execution in Complex Dynamic Worlds. PhD thesis, Yale University (1989)
4. Newell, A.: Unified Theories of Cognition. Harvard University Press (1990)
5. Anderson, J.R., Matessa, M., Lebiere, C.: Act-r: A theory of higher level cognition and its relation to visual attention. HUMAN-COMPUTER INTERACTION **12** (1997) 430–462
6. Firby, R.J.: Task networks for controlling continuous processes. In: the Second International Conference on AI Planning Systems. (1994)
7. Ishida, T., Fukumoto, M.: Interaction design language q : The proposal(in japanese). In: Proceeding of JSAI'01. (2001)
8. Ishida, T.: Q: A scenario description language for interactive(to appear). IEEE Computer (2002)
9. Nakashima, H., Noda, I., Handa, K.: Organic programming language gaea for multi-agents. In: Proceedings of International Conference on Multi-Agent Systems 96, AAAI Press (1996) 236–243
10. Nakashima, H., Noda, I.: Dynamic subsumption architecture for programming intelligent agents. In: Proceedings of International Conference on Multi-Agent Systems 98, AAAI Press (1998) 190–197
11. Noda, I., Takahashi, T., Morita, S., Koto, T., Tadokoro, S.: Language design for rescue agent. In: Proceedings of RoboCup 2001 Symposium. (2001)