Coordination of Concurrent Scenarios in Multi-Agent Interaction

Rie Tanaka, Hideyuki Nakanishi, and Toru Ishida

Department of Social Informatics, Kyoto University
Yoshida Honmachi, Sakyoku, Kyoto, 6068501, Japan
+81-75-753-4820
rtanaka@ai.soc.i.kyoto-u.ac.jp

Abstract. Though research on agents that interact with humans via voice or text has been intensively conducted, agents that can interact with more than two persons in parallel have not been studied well. To enable an agent to interact with multiple people, we propose a method to assign multiple scenarios to one agent, in which each scenario describes an interaction protocol between the agent and one person. Obviously, coordination among multiple scenarios is required to avoid conflicts in actions. For example, one agent cannot execute both walking and sitting actions simultaneously. However, what is more important is that a coordination policy, a way of specifying how to manage conflicts among multiple actions, must be introduced. In this paper, we introduce a coordination scenario that avoids conflicts in actions and coordinates scenarios according to a coordination policy. The coordination scenario, which controls the execution of interaction scenarios by communication, is generated by a coordination policy for solving conflicts. When the coordination scenario receives a request to execute actions from an interaction scenario, it checks whether the actions will trigger conflicts and sends an order not to execute them if conflicts will occur. The coordination scenario interworks concurrent interaction scenarios by executing this process repeatedly.

Keywords. Agent and digital cities; Conflict resolution and negotiation; Agent communication languages, dialog & interaction protocols; Multiagent systems and their applications.

1 Introduction

Many studies on agents interacting with humans by voice or text have been conducted. In studies on embodied conversational agents[Cassell 00], agents embodied in a virtual space that use verbal and nonverbal communications have been constructed. For example, Cosmo[Lester 00], a life-like pedagogical agent, teaches about networks one-on-one by pointing to objects, while Steve[Rickel 99b], a task-oriented pedagogical agent, interacts with avatars controlled by users and achieves general tasks in the virtual world.

This research aims to enable an embodied agent to successfully interact with multiple people by verbal and nonverbal communication. We note that not all interactions in the real world are just one-on-one, and the purposes of interaction or the roles in interaction are not always obvious. Humans have different ways of interacting with different persons; we call them by the general term interaction protocols. When interacting with multiple people, protocols are chosen to suite each person. Therefore, in this paper, rather than plural plans or rules, we give plural protocols to each agent to achieve the entire goal. We control agents by scenarios, which describe the interaction protocols in an executable form.

To merge multiple protocols into single scenario is difficult. This is because all combinations of all possible behaviors of each person should be described for the purpose of enabling the agent to interact with multiple persons successfully even if they behave individually, i.e. not cooperatively. Therefore, we describe one protocol in each scenario and execute the scenarios concurrently. Since problems occur when we execute multiple scenarios concurrently, we propose a method that coordinates concurrent scenarios so as to execute them successfully.

2 Problems with concurrent execution

Assume that a sales agent has scenario Sa for interaction with customer Ca and scenario Sb for interaction with customer Cb. Interaction which occurs between the agent and customer Ca and between the agent and customer Cb is shown

⟨ Interaction using scenario Sa ⟩
Customer Ca : Excuse me.
Sales agent : May I help you?
Customer Ca : Do you have umbrellas?

⟨ Interaction using scenario Sb ⟩
Customer Cb : Excuse me.
Sales agent : May I help you?
Customer Cb : Could you tell me the way to Kyoto-Station?

Fig. 1. Examples of interaction with a single person

⟨ Interaction pattern 1 ⟩
Customer Ca : Excuse me.
Sales agent : May I help you?
Customer Cb : Excuse me.
Sales agent : May I help you?
Customer Ca : Do you have umbrellas?
Customer Cb : Could you tell me the way to Kyoto-Station?

⟨ Interaction pattern 2 ⟩
Customer Ca : Excuse me.
Sales agent : May I help you?
Customer Cb : Excuse me.
Sales agent : Please wait a minute.
Customer Ca : Do you have umbrellas?
Sales agent : (response)
(Interaction with customer Ca finishes.)
Sales agent : Thank you for waiting.
Customer Cb : Could you tell me the way to Kyoto-Station?

Fig. 2. Examples of interaction with multiple persons

in Figure 1. When customers Ca and Cb contact the agent, there are several interaction patterns (two of them are shown in Figure 2). As shown by these examples, there are various ways of interacting with multiple persons.

To realize the interaction shown in Figure 2, it is necessary to solve two problems. One problem is the conflicts in actions. For example, the sales agent tries to speak to customers Ca and Cb simultaneously if they approach the agent at the same time. Since the sales agent has only one body, such an action is physically impossible. The other problem is that we need to design a coordination policy to coordinate scenarios. A coordination policy specifies how to manage conflicts, in other words, how to coordinate scenarios. In the above example, the coordination policy would specify which person the agent responds first. It is necessary to define the method to describe the coordination policy and reflect it to the coordination scenario.

3 Scenario coordination architecture

3.1 Coordination scenario

This research assumes that each agent has several scenarios. In related research, in the area of multi-agent planning, Georgeff suggested the method of coordination among agents, each of whom has a plan[Georgeff 83]. He suggested that communication functions are added to each plan, and plans are coordinated by the synchronization program. In this research, we propose the use of a coordination scenario to coordinate interaction scenarios.

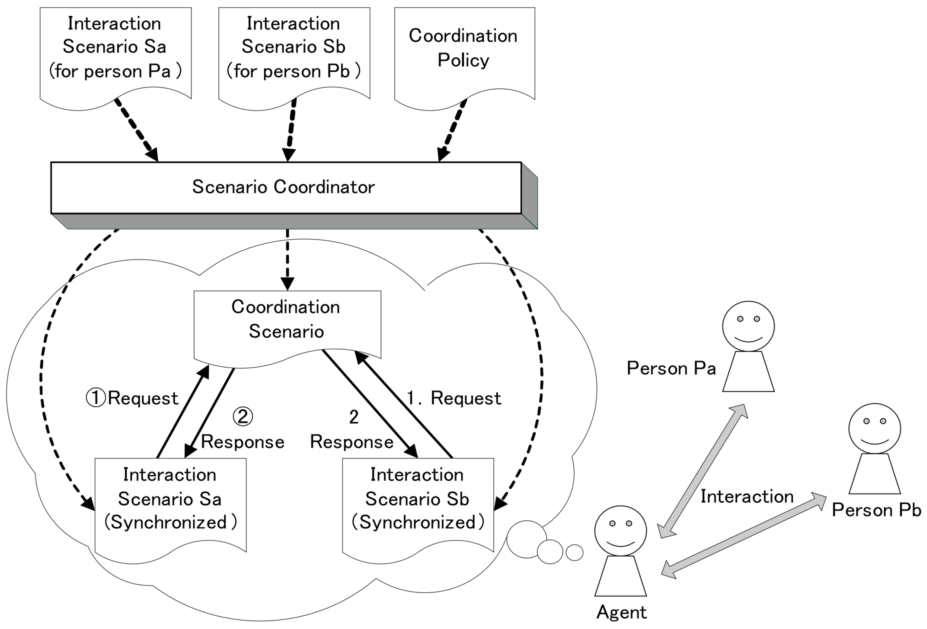
Interaction Scenario Sa (for person Pa)
Interaction Scenario Sb (for person Pb)
Coordination Policy
Scenario Coordinator
Coordination Scenario
①Request
②Response
2 Response
1. Request
Interaction Scenario Sa (Synchronized)
Interaction Scenario Sb (Synchronized)
Person Pa
Person Pb
Interaction
Agent

Fig. 3. Scenario coordination architecture

The coordination scenario controls the execution of actions by communicating with each interaction scenario. A protocol consists of pairs of an event and corresponding sequential multiple actions, so the coordination scenario controls action sequences. The coordination scenario obeys the coordination policy described by the scenario writer. Furthermore, it has functions of detecting and solving conflicts in the action sequences. The coordination scenario examines whether conflicts can occur, and if conflict is possible, it orders interaction scenarios not to execute the offending action sequences.

Figure 3 shows the whole process of coordination. The scenario coordinator automatically generates the coordination scenario from the coordination policy and interaction scenarios. It also changes the interaction scenarios so as to communicate with the coordination scenario. Every time an event is observed and actions corresponding to it are to be executed in the changed interaction scenario, the coordination scenario controls the scenario in the following manner. 1)Changed interaction scenarios send requests to the coordination scenario before executing an action sequence and wait for a response. 2)The coordination scenario examines whether a requested action sequence conflicts with the action sequences being executed, and gives an order to execute it only if it doesn't conflict with any of action sequences. 3)Each interaction scenario obeys the response of the coordination scenario.

3.2 Scenario coordinator

The scenario coordinator changes the interaction scenarios if necessary and generates the coordination scenario. As for the interaction scenarios, the scenario coordinator adds communication functions to each scenario which describes one-on-one interaction.

The coordination scenario is generated by the scenario coordinator from both the coordination policy and interaction scenarios. At first, the scenario coordinator examines all combinations of the action sequences used in all interaction scenarios and if pairs of action sequences are found to conflict, the pairing is stored as conflict data. When the scenario coordinator detects conflicts in action sequences, it refers to the definition of conflict conditions, which we define as follows. The scenario coordinator refers to the coordination policy when generating the coordination scenario so that the functions in the coordination scenario refer to the conflict data.

We define conflict conditions using precondition, postcondition, continuance condition, and the resource of the action sequence. The continuance condition should be satisfied during execution of an action sequence that takes some time, for example, talking action or walking action. The resource means, for example, the legs or hands of the agent. There are five conflict conditions as follows. The scenario coordinator judges that conflict will occur if any one of the conditions is satisfied.

1) The concurrent execution of the action sequences yields a result that differs from the result of any sequential execution.
2) Continuance conditions of the action sequences contradict each other.
3) The continuance condition of an action sequence currently being performed makes it impossible to satisfy the precondition of the next action sequence.
4) The continuance condition of one action sequence contradicts the postcondition of the other.
5) The action sequences use the same resources.

4 Scenario coordination mechanism

4.1 Generation of the coordination scenario

We describe the coordination policy in the same format as the interaction scenarios. The reason is as follows. The coordination scenario coordinates interaction scenarios by controlling execution of the actions through communication. In other words, the coordination scenario repeats the process of receiving a request message, such as a request to execute an action sequence, from an interaction scenario and sending a response message to the request. Since interaction scenarios describe observations of the environment and actions corresponding to the observation results, this format is suitable for the coordination policy.

In the coordination policy, request messages sent by interaction scenarios are the observed target, and the sending of response messages from the coordination

Table 1. The meanings of messages

Request Messages	Meanings
execute request	Order to execute an action sequence
action end report	Report the end of an action sequence
wait report	Report the end of the before-wait procedure
scenario end report	Report the end of execution of a scenario

Response Messages	Meanings
execute instruction	Order to execute an action sequence
annul instruction	Order not to execute an action sequence
wait instruction	Order not to execute an action sequence and execute the before-wait procedure

scenario is equivalent to the actions corresponding to the observation results. Table 1 shows typical request and response messages. In the messages, the wait report and the wait instruction are used in the wait procedure, which is initiated when the agent wants to keep a person waiting. The wait procedure is executed as follows. 1)The coordination scenario sends a wait instruction to the interaction scenario associated with the person who is to be kept waiting. 2)The interaction scenario executes the before-wait procedure, for example, saying "Please wait a minute" to inform the person of the agent's intention. 3)The interaction scenario sends a wait report to the coordination scenario and waits for the order to resume. When the interaction scenario receives an execute instruction, which means the order to resume, it executes the after-wait procedure, for example, saying "Thank you for waiting". After execution of the after-wait procedure, it proceeds with the interaction. In this process, the behaviors used in the before-wait procedure and the after-wait procedure can be freely set.

Figure 4 shows the coordination policies following the two patterns in Figure 2. It is described by using the state machine model. The policy corresponding to pattern 1 means that the coordination scenario, which is generated from the policy, sends an execute instruction every time an action sequence is requested without consideration of which scenario sent the request.

The policy corresponding to pattern 2 means that the agent keeps a person waiting until the current interaction finishes. Specifically, since one scenario describes interaction with one person, the coordination scenario sends an order to the second scenario to wait until execution of the first scenario finishes. At first, when a person coarrives and the corresponding scenario sends an execute request, the coordination scenario sends an execute instruction and transits to the reception state. When the coordination scenario receives an execute request, it sends an execute instruction to the scenario being executed, or a wait instruction to any other scenario. After execution of the scenario finishes, the coordination scenario transits to the preparing state. If it receives a wait report, which means

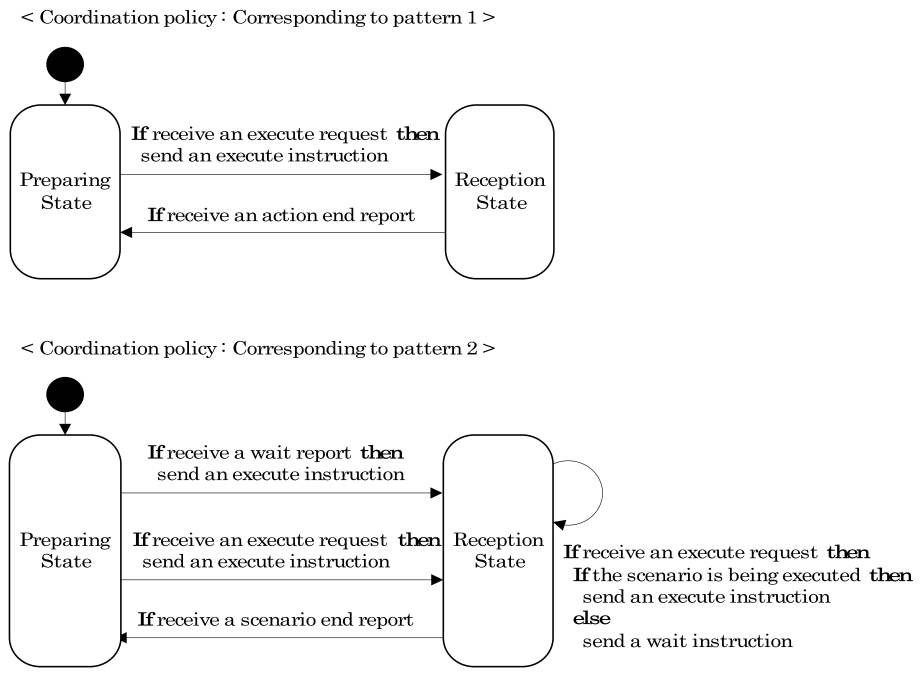
< Coordination policy : Corresponding to pattern 1 >
Preparing State
Reception State
If receive an execute request **then** send an execute instruction
If receive an action end report
< Coordination policy : Corresponding to pattern 2 >
Preparing State
Reception State
If receive a wait report **then** send an execute instruction
If receive an execute request **then** send an execute instruction
If receive a scenario end report
If receive an execute request **then**
If the scenario is being executed **then**
send an execute instruction
else
send a wait instruction

Fig. 4. Examples of the coordination policy

there are one or more scenarios waiting, it sends an execute instruction to resume the execution and transits to the reception state. The coordination scenario continually repeats this process.

The coordination scenario is generated simply by changing the observation parts and action parts in the coordination policy into executable functions. However, since no description of the conflicts is included in the coordination policy, the scenario coordinator generates the coordination scenario by adding functions for examining the conflicts to the changed coordination policy. Functions are added to the part that sends the execute instructions. Figure 5 shows the algorithm used to send an execute instruction. There are two kinds of situations in which an execute instruction is sent. One is the case when the coordination scenario sends order corresponding to the execute request. The other is the case when it sends order to the scenario which keeps waiting for the execute instruction to resume interaction. In the latter case, the coordination scenario keeps waiting during the action sequence to be resumed conflicts with the action sequences being executed. This is because, the scenario which has sent a wait report keeps waiting for an execute instruction and it is impossible to annul the resumption.

```
$x \leftarrow$ the action sequence
S $\leftarrow$ list of action sequences being executed

**If** the execute instruction corresponds to the execute request **then**
  **If** S is empty **then** send an execute instruction
  **else**
    **for each** $s$ **in** S **do**
      **if** $x$ conflicts with $s$ **then**
        send an annul instruction
        **return**
    **end**
      send an execute instruction
**else if** the execute instruction corresponds to the wait report **then**
  **If** S is empty **then** send an execute instruction
  **else**
    result $\leftarrow$ *false*
    **loop do**
      **for each** $s$ **in** S **do**
        **if** $x$ conflicts with $s$ **then** result $\leftarrow$ *true*
      **end**
      **if** result $=$ *false* **then**
        send an execute instruction
        **return**
    **end**
```

Fig. 5. Algorithm to send an execute instruction

4.2 Synchronization of interaction scenarios

The scenario coordinator adds functions of sending request messages to the coordination scenario and following response messages to each interaction scenario. The interaction scenario consists of several rules. Figure 6 shows the algorithm used to add functions to each rule. Figure 7 shows the rule after adding functions in accordance with the algorithm. If an interaction scenario receives an execute instruction, it simply obeys the instruction. However, if it receives an annul instruction, it not only obeys it, but also memorizes that the event was observed. After that, if it observes a new event, it sends a request to execute the new action sequence. If no new event is observed, it sends a request to execute the action sequence corresponding to the memorized event again.

By following the above procedure, an interaction scenario can repeatedly send the same request until it receives the execute instruction, except the case when the situation has changed. If the situation has changed, namely, if it is needed to execute new action sequence corresponding to new event instead of memorized event, the interaction scenario can annul the memorized event and send a new request. This indicates that the interactions scenario can meet the change of situations.

The interaction scenario consists of several rules. We represent the rule as below.

If *event* is observed **then** execute *actions*

We call the part "**If** *event* is observed **then**" as the conditional part, and the part "execute *actions*" as the action part.

The scenario coordinator loads a scenario from a file.

```
S ← scenario
D ← empty list
for each rule in S do
  L, C ← empty list
  Add character strings which mean following commands or part to the end of C
     A command "send an execute request to the coordination scenario"
     A command "wait for response"
     A conditional part "If the response = an execute instruction then"
     The action part of rule
     A command "send an action end report to the coordination scenario"
     A conditional part "else if the response = a wait instruction then"
     A command "execute the before-wait procedure"
     A command "send a wait report to the coordination scenario"
     A command "wait for an execute instruction"
     A command "execute the after-wait procedure"
     The action part of rule
     A command "send an action end report to the coordination scenario"
     A conditional part "else if the response = an annul instruction then"
     A command "memorize that event was observed"
     A command "send an action end report to the coordination scenario"
     A command "continue observation"
  Add the conditional part of rule to the end of L
  Add C to the end of L
  Add a character string which means a conditional part
  "else if it is memorized that event was observed then" to the end of L
  Add the copy of C to the end of L
  Add L to the end of D
end
Write each element of the list D into a new file in order
```

Fig. 6. Algorithm to add functions to interaction scenarios

```
If event is observed then
  Send an execute request to the coordination scenario
  Wait for response
  If the response = an execute instruction then
    Execute actions
    Send an action end report to the coordination scenario
  else if the response = a wait instruction then
    Execute the before-wait procedure
    Send a wait report to the coordination scenario
    Wait for an execute instruction
    Execute the after-wait procedure
    Execute actions
    Send an action end report to the coordination scenario
  else if the response = an annul instruction then
    Memorize that event was observed
    Send an action end report to the coordination scenario
    Continue observation
else if it is memorized that event was observed then
  Execute the same procedure executed when event was observed
```

Fig. 7. Synchronization of interaction scenarios

5 Implementation

We realized our proposal by using FreeWalk/*Q*[Nakanishi 04]; it allows us to control agents by scenarios which describe interaction protocols. In FreeWalk/*Q*, protocols are described in *Q* language, and are called *Q* scenarios. To describe a scenario, we use defined cues which mean an event that triggers interaction and actions which are behaviors corresponding to cues. Therefore, we used the state machine model to describe the coordination policy as well as scenarios, and defined cues and actions that are used to describe the coordination policy. We implemented the scenario coordinator which generates the coordination scenario and changes the interaction scenarios as described above.

Figure 8 displays a screen shot of FreeWalk/*Q*. The woman at the center is a sales agent and has the coordination scenario and changed scenarios. We use the coordination policy corresponding to pattern 2, which is shown in Figure 4. During the execution of scenarios, she first interacts with the male customer on the right. When the female customer attempts to talk her, she says "Please wait a minute" and keeps her waiting. When the interaction with the male customer finishes, she turns to the female customer and says "Thank you for waiting."

6 Conclusion

This research aims to enable agents to successfully interact with multiple people. Specifically, we assume that multiple interaction protocols are assigned to

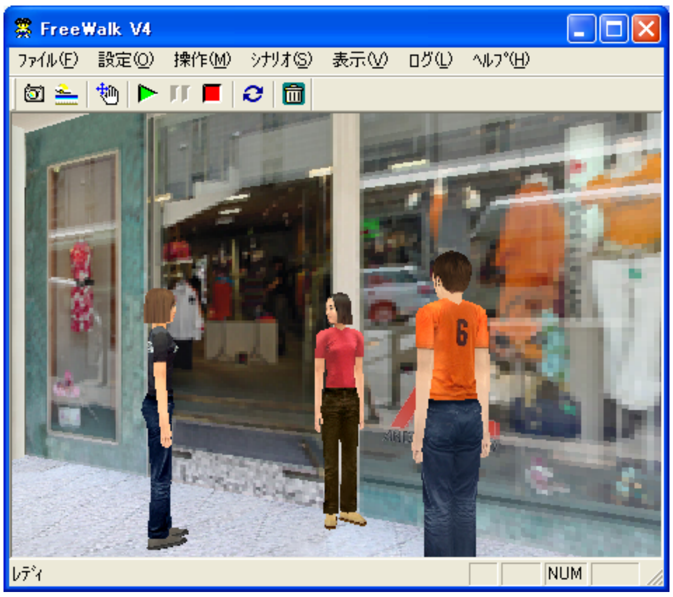

Fig. 8. Screen shot of FreeWalk/*Q*

one agent. We design one interaction scenario for each interaction protocol and assign several interaction scenarios to one agent. The agent executes scenarios concurrently. If we execute multiple scenarios concurrently, we must prevent conflict between the multiple actions, and a coordination policy that describes how to manage conflicts is needed. For coordination, we introduce the coordination scenario; it communicates with the interaction scenarios to control the execution of actions.

The coordination scenario is generated by adding functions of detecting and solving conflicts to the coordination policy. We propose a scenario coordination architecture, in which the coordination scenario is generated and interaction scenarios are changed so as to communicate with the coordination scenario automatically. Generated scenarios are assigned to the agent, and the coordination scenario coordinates the interaction scenarios at run time. Furthermore, we implemented the architecture and verified that the agent behaves according to the coordination policy set.

In this paper, we propose a scenario coordination mechanism for concurrent execution. At run time, the interaction scenario sends a request to execute actions to the coordination scenario, and the coordination scenario sends orders to the interaction scenarios to execute them or not. The coordination scenario determines if the actions can be executed or not by referring to the five types of conflicts which we have newly defined.

The method proposed in this paper allows us to describe a policy without knowing the contents of interaction scenarios in detail. The innovation of this research is that interaction scenarios described by different persons can now be coordinated quite easily.

Acknowledgement

This work was supported by a JSPS Grant-in-Aid for Scientific Research (17650041) and the International Communication Foundation.

References

[Cassell 00] Cassell, J., Sullivan, J., Prevost, S. and Churchill, E.: *Embodied Conversational Agents*, MIT Press (2000).

[Lester 00] Lester, J. C., Towns, S. G., Callaway, C. B., Voerman, J. L. and FitzGerald, P. J.: Deictic and Emotive Communication in Animated Pedagogical Agents, In Cassell, J., Sullivan, J., Prevost, S. and Churchill, E.: *Embodied Conversational Agents*, MIT Press, pp. 123–154 (2000).

[Rickel 99b] Rickel, J. and Johnson, W. L..: Virtual Humans for Team Training in Virtual Reality, In *Proceedings of the Ninth International Conference on AI in Education*, pp. 578–585, IOS Press (1999b).

[Nakanishi 04] Nakanishi, H. and Ishida, T.: FreeWalk/*Q*: Social Interaction Platform in Virtual Space, *ACM Symposium on Virtual Reality Software and Technology (VRST2004)*, pp. 97–104 (2004).

[Ishida 02] Ishida, T.: *Q*: A Scenario Description Language for Interactive Agents, *IEEE Computer*, Vol. 35, No. 11, pp. 54–59 (2002).

[Georgeff 83] Georgeff, M.: Communication and interaction in multiagent planning, In *Proceedings of the 3th National Conference on Artificial Intelligence*, pp. 125-129 (1983).