A constructive approach for developing interactive humanoid robots

Takayuki Kanda[1], Hiroshi Ishiguro[1,2], Michita Imai[1], Tetsuo Ono[1,3], and Kenji Mase[1]

[1]ATR Media Information Science Research Laboratories, Kyoto, Japan, {kanda,michita,mase}@atr.co.jp
[2]Wakayama University, Wakayama, Japan, ishiguro@sys.wakayama-u.ac.jp
[3]Future University-Hakodate, Hokkaido, Japan, tono@fun.ac.jp

Abstract

There is a strong correlation between the number of appropriate behaviors an interactive robot can produce and its perceived intelligence. We propose a robot architecture for implementing a large number of behaviors and a visualizing tool for understanding the developed complex system. Behaviors are designed by using knowledge obtained through cognitive experiments and implemented by using situated recognition. By representing relationships between behaviors, episode rules help to guide the robot in communicating with people in a consistent manner. We have implemented over 100 behaviors and 800 episode rules in a humanoid robot. As a result, the robot could entice people to relate to it interpersonally. An Episode Editor is a tool to support the development of episode rules and to visualize the complex relationships among the behaviors. We consider the visualization is to be necessary for the constructive approach.

1. Introduction

Recent progress in robotics research has brought with it a new research direction, “interaction-oriented robots.” These robots are different from traditional task-oriented robots, such as industrial robots, which perform particular tasks in limited areas. The interaction-oriented robots are designed to communicate with humans and will be able to participate in human society. We are trying to develop such an interaction-oriented robot that will exist as a partner in our daily life. As well as performing physical support functions, these robots will act as a new form of media for information communications.

Several researchers are endeavoring to realize such interaction-oriented robots [1,2]. Since an interactive robot's subsystems are evaluated in the context of the total system (both robot and environment), it is important not only to develop subsystems (recognition, learning, and so on), but also to build up complete robotic systems that interact with humans and work in real environments. It is especially important to develop a large number of behaviors. The complex relationships among these behaviors let the robot give an intelligent impression to humans and encourage humans to entrain into interactions with the robot. Our constructive approach is to continue to implement behaviors until humans think the robot has an animated and lifelike existence beyond that of a simple automatic machine. After implementing 40 behaviors and 300 relationships among them in a humanoid robot that had enough sensors and physical capacity to express itself, the robot got people to relate to it interpersonally [3]. Such interpersonal behaviors are also observed in the interaction between a child and the robots [4]. Now, the number of behaviors is over 100. We believe this is a starting point to discuss the robots’s intelligence and the mechanisms. For example, we consider it important for these robots to interact with other robots as well [5].

Regarding the design of how the robot interacts, we consider that the active interaction approach is suitable for supplementing imperfect sensory processing technologies. Ono and his colleagues discussed the importance of bi-directional communication [6]. That is, robots do not simply obey the commands of humans but communicate with them as equals. In addition, current sensory-recognition technology is not sufficient to recognize every human behavior. Rather, in our approach, robots are proactive in initiating interactions and entice humans adaptively respond to their actions. A robot's particular embodiment (head, eyes, arms, etc.) helps the active interaction to entrain humans.

Although such an active interaction approach works effectively for a while in certain situations, it lacks mechanisms for maintaining a consistent context for long-term interaction. For example, if a robot asks a human “Where are you from?” and the robot asks the same question a few minutes later, this is obviously wrong. We propose a rule-based constraint mechanism in addition to active interaction. Traditionally, a production system such as [7,8] is a popular rule-based approach.

In this paper, we propose an architecture for interaction-oriented robots that have a large number of behaviors. The interactive behaviors are designed with knowledge about the robot’s embodiment obtained from cognitive experiments, and then implemented as *situated modules* with situation-dependent sensory data processing for understanding complex human behaviors. The relationships between behaviors are implemented as rules governing execution order (named “*episode rules*”) to maintain a consistent context for communication. The implemented 102 *situated modules* and 884 *episode rules* generate a complicated switching of behaviors. A development tool named “*Episode Editor*” supports our constructive approach by expressing the complex relationships and the execution of many simple behaviors visually.

2. Robot Architecture Based on Episode Rules for Realizing Interaction with Humans

We propose an architecture for interaction-oriented robots that autonomously interact with humans. The basic components of the system are *situated modules* and *episode rules*. The robot system sequentially executes *situated modules*, and *episode rules* govern their execution order. This is an extension of our previous architecture [3], which has merits for the development and use of body properties

The basic strategy of implementation is as follows:

1. Develop *situated modules* for various situations.
2. Define the basic execution order of *situated modules* with *episode rules* for sequential transition.
3. Add *episode rules* for reactive transitions.
4. Modify implemented *episode rules*, and specify *episode rules* of negation to suppress execution of *situated modules* for a particular long-term context.

The visualization support function of the *Episode Editor* will be helpful especially at step 4.

2.1. Situated Modules

In linguistic research, an adjacency pair is a well-known term for a unit of conversation where the first expression of a pair requires the second expression to be of a certain type (greeting and response, question and answer, and so on.) Similarly, we consider that human-robot interaction can be mainly realized with action-reaction pairs. That is, when a human acts toward the robot, it reacts to the human's action; when the robot acts toward a human, the human reacts to its action.

The *situated module* realizes the action-reaction pair as an interactive and reactive behavior (indication-recognition pair) in a particular situation. With the active interaction approach, the robot mainly acts and humans react by sequential transition. Deviation from the basis is treated by reactive transition and *reactive modules*. The robot executes just one *situated module* at a time.

Precondition, Indication, and Recognition Parts

Each *situated module* consists of precondition, indication, and recognition parts (Fig. 1) and is implemented using *communicative units*. By executing the precondition, the robot system checks whether the *situated module* is in an executable situation or not. For example, the *situated module* that realizes talking about the weather by retrieving weather information from the Internet is not executable (precondition is not satisfied) when the robot system cannot connect to the Internet.

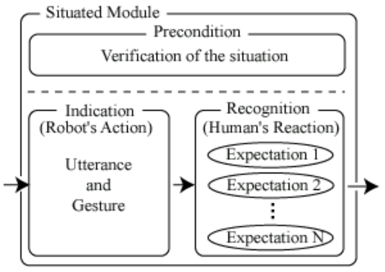
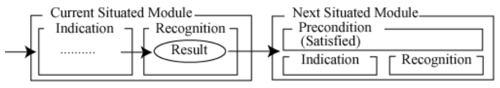
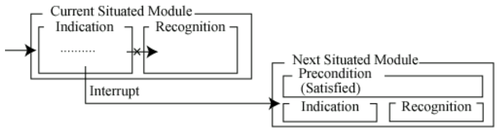


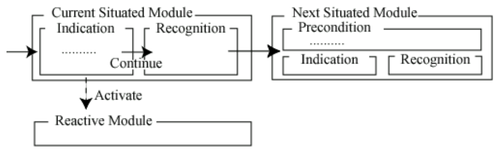
***Figure 1**: Situated module*



(a) Sequential transition (human reacts to the robot)



(b) Reactive transition (the robot reacts to human's interruption)



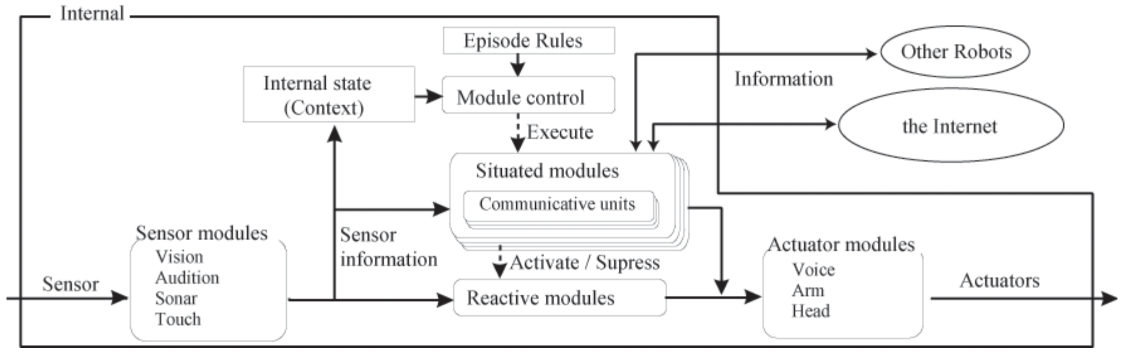
(c) Activation of Reactive Modules (robot reacts; no transition)

***Figure 2**: Transition of situated modules*

By executing the indication part, the robot takes an action to interact with humans. For example, for the handshake module, it says "Let's shake hands" and offers its hand.

The recognition part is designed to recognize several kinds of human reactions towards the robot's action caused by the indication part. That is, it is an expectation of the human's reaction. The *situated module* itself produces the particular situation, and then it can recognize complex human behaviors under the particular situation.

A Communicative unit is a sensory-action unit that realizes a basic action for natural and effective human-robot communications. Each *communicative unit* is retrieved from experiments concerning the kinds of interactions the



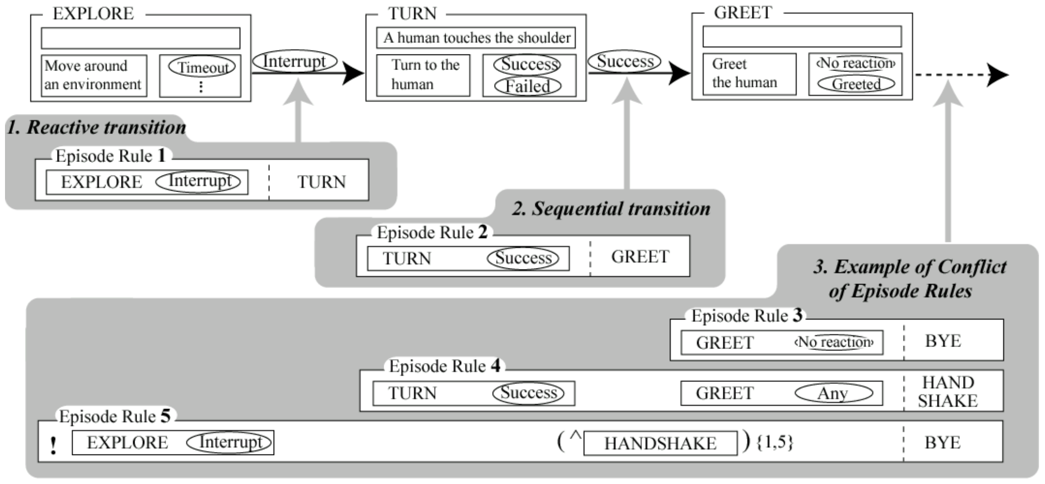


***Figure 4**: Illustrated example of the transitions of the current situated modules ruled by episode rules*

***Table 1**: Grammar of episode rules*

1. <ModuleID=result_value>…<…>NextModule
2. (<ModuleID1=result_value1>|<ModuleID2=result_value2>)...
3. (…){n,m}…
4. !<…>NextModule
5. ^<ModuleID=^result_value>NextModule

(1:basic structure of describing executed sequence,
2: “OR”, 3: repetitions, 4: negation of *episode rule*,
5: negation of Module ID and result value)

robot’s body affords it. Concretely, we have implemented ‘gaze object’, ‘eye contact’, ‘nod’, and so forth. *Situated module* is implemented by coupling *communicative units* with directly supplementing other sensory-action units (particular utterance, positional movement and so on.)

Sequential and Reactive Transition of *Situated Modules*, and *Reactive Modules*

After the robot executes the indication part of the current *situated module*, it recognizes the human’s reaction by the recognition part. Then it ends the execution of the current *situated module*, records the result value corresponding to the recognition result, and transits to the next executable *situated module* (Fig. 2 (a)). The next module is decided on by the result value of the current *situated module* and the execution history of *situated modules*. This sequential transition is ruled by *episode rules*.

Thus, the human-robot communication for a consistent context can be realized by sequential transition. However, there are important problems of the communication, which are unresolved by sequential transition: interruption and deviation. For example, when some humans talk to each other and suddenly a telephone rings, they will stop the conversation to respond to the telephone call. Such an interruption and deviation is dealt with reactive transition and *reactive modules*. Reactive transition is also ruled by *episode rules* (Fig. 2 (b)). If the reactive transition is assigned for the current situation and the precondition of the corresponding next *situated module* is satisfied, the robot quits executing the current *situated module* and instantly transits to the next *situated module*.

Reactive modules are also prepared for an interruption, but in this case the robot does not quit the execution of the current *situated module* when *a reactive module* is activated (Fig. 2 (c)). Instead, the robot system executes the *reactive module* in parallel with the current *situated module*. For example, we implemented a *reactive module* to make the robot gaze at the part of its body being touched. When the robot talks to a human and the human suddenly touches the arm of the robot, the robot gazes at the arm to indicate that it has noticed it, but continues talking. Similar to the subsumption architecture [9], upper hierarchy modules (*situated modules*) can suppress lower ones (*reactive modules*).

Other Components of the Architecture

The architecture has components for communication using computer networks. This is a new information infrastructure that lets robots keep humans informed by communicating with them in natural language. For example, when the robot and humans talks about weather, the robot obtains weather information from *the Internet.* If the forecast is raining, it says, “It will rain tomorrow.” *Internal state* represents the context based on the execution history of the *situated modules*. Inputs from sensors are pre-processed at *sensor modules* such as speech recognition. *Actuator modules* perform low-level control of actuators.

2.2. Episode Rules

Episode rules guide the robot into a new episode of interaction with humans by controlling transitions between *situated modules*. All *episode rules* are compared with the current *situated module* and the execution history of *situated modules* to determine which situated module to execute next. The system performs the comparison in the background of the current *situated module’*s execution and prepares the next executable module list. After the current module’s execution, the robot checks the preconditions of each *situated module* in the list. If the precondition is satisfied, it transits to the next *situated module*. Each *episode rule* has a priority. If some *episode rules* conflict, the *episode rule* with higher priority is used.

Table 1 indicates the basic grammar of the *episode rule*. Each *situated module* has a unique identifier called a *Module ID*. "<Module ID = result value>" is the rule to refer to the execution history and the result value of the *situated modules*, then "<ModuleID1=result value 1>< ModuleID2=result value 2>⋯" means a referring rule of previously executed sequence of *situated modules* (Table 1-1). "<…>|<…>" means a selective-group (*OR*) of the executed *situated modules*, then "(…)" means the block that consists of a *situated module*, a sequence of *situated modules*, or a selective-group of *situated modules (*Table 1-2). Similar to the regular expression, we can describe the repetition of the block as "(⋯){*n*,<u>*m*</u>}", where *n* gives the minimum number of times to match the block and *m* gives the maximum (Table 1-3). We can specify the negation of the whole episode rule with an exclamation mark "!". For example, "!<…>…<…>NextModuleID" (Table 1-4) means the module of NextModuleID will *not* be executed when the *episode rule* matches the current situation specified by "<…>…<…>". The negation of ModuleID and result value can be written as caret character "^" (Table 1-5).

Figure 4 is an example of a transition. At first, the robot explores the environment by "EXPLORE." Then, a human touches the shoulder of the robot. This action causes a reactive transition ruled by *episode rule* 1 (Fig. 4-1). The robot turns to the human by executing the *situated module* "TURN." After the execution of "TURN," it starts to greet him/her by executing "GREET." This sequential transition is caused by *episode rule* 2 (Fig. 4-2).

Next, we explain conflict of *episode rules* (Fig. 4-3). When "GREET" results in "No reaction," "BYE" is the candidate of the next *situated module* selected by *episode rule* 3. Meanwhile, "HANDSHAKE" is the candidate selected by *episode rule* 4. *Episode rule* 5 is a negative *episode rule* to suppress the transition to "BYE" (it specifies that the robot should not say goodbye before the handshake once its exploration has been interrupted). If the priority of *episode rule* 5 is higher than that of *episode rule* 3, "BYE" is not the candidate of the next execution.

2.3. Episode Editor

We developed the *Episode Editor*, which visually displays the complex relationships and execution of many simple situated modules. This enables us to intuitively develop a large number of *situated modules* and relationships among them (*episode rules*). The *Episode Editor* has three main functions.

Implementation of *Episode Rules*

The *Episode Editor* has the obvious function of editing *episode rules*. In Figure 5, half of the left of the screen displays the *episode rule* that a developer is editing. Each box indicates one block of *situated modules*. If there is a complex block assigned by "(…)", the block is expanded in the left-second box of the screen. The right-most box on the left screen indicates the next executable *situated module*. The right half of the screen shows the detail of the selected box in the left screen, where developers can indicate the identifier of the *situated modules* (Module ID), result values, repetition, and negation. This editing screen is opened from the main screen (Fig. 6), which has two windows: a list of all implemented *episode rules* and the search result of *episode rules*. Developers can select one of the *episode rules* from the lists to edit it.

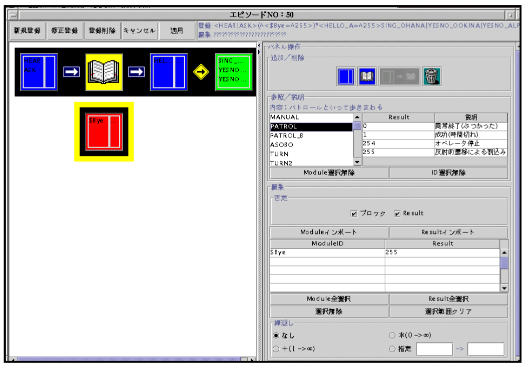


Figure 5*: Editing screen of the Episode Editor: developing or modifying an episode rule*

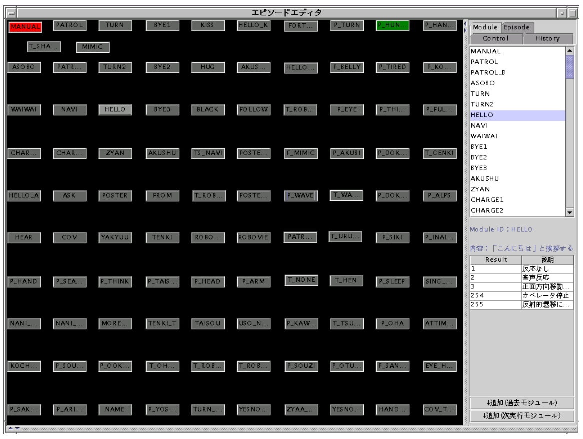
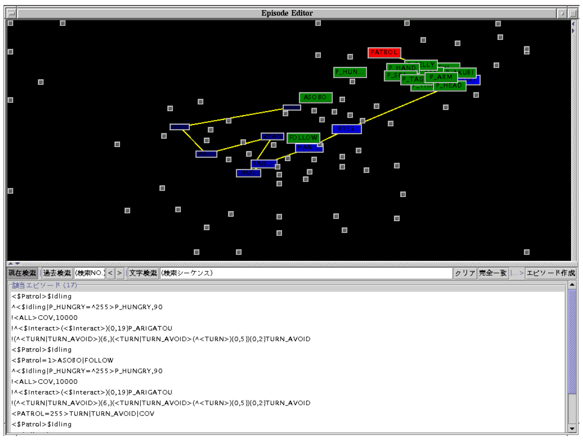


Figure 6*: Main screen of the Episode Editor: all situated modules (gray boxes) are displayed in rows and lines before calculation of positions.*



該当エピソード (17)

```
<$Patrol>$Idling
^<$Idling|P_HUNGRY=^255>P_HUNGRY,90
!<ALL>COV,10000
!^<$Interact>(<$Interact>){0,19}P_ARIGATOU
!(^<TURN|TURN_AVOID>){6,}(<TURN|TURN_AVOID>(^<TURN>){0,5}){0,2}TURN_AVOID
<$Patrol>$Idling
<$Patrol=1>ASOBO|FOLLOW
^<$Idling|P_HUNGRY=^255>P_HUNGRY,90
!<ALL>COV,10000
!^<$Interact>(<$Interact>){0,19}P_ARIGATOU
!(^<TURN|TURN_AVOID>){6,}(<TURN|TURN_AVOID>(^<TURN>){0,5}){0,2}TURN_AVOID
<PATROL=255>TURN|TURN_AVOID|COV
<$Patrol>$Idling
```

Figure 7*: Displaying the current status and search result. upper screen displays the current status, lower screen shows the search result of episode rules*

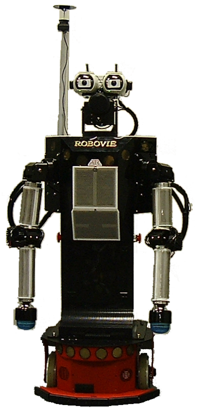
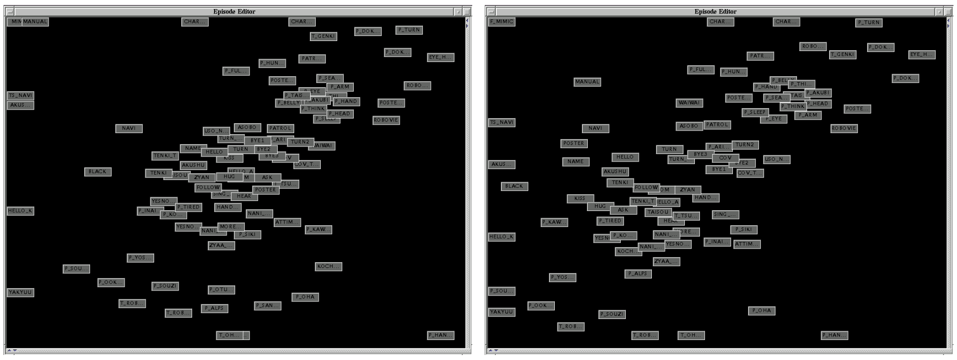


Figure 8: *"Robovie"*



Subject 1 *Subject 2*

Figure 9: *Results of the visualization by the Episode Editor*

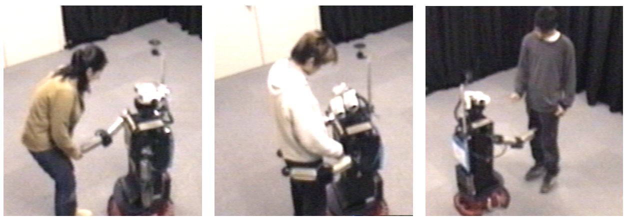


Figure 10: *Implemented playing behaviors*

Visualization of *Situated Modules* and *Episode Rules*

The *Episode Editor* has two types of visualization support functions for past episodes: the position of situated modules for all execution history, and the size and color for the recent history and current state. The *situated modules* are placed according to their relationships (Fig. 7), which are calculated from the execution history. For example, if *situated module* X is frequently executed after Y, X and Y have a strong relationship, and appear near to each other in the main screen. The force of this relationship between two *situated modules* is calculated by the following formula, which is known as the spring model method:

$$F_{ij} = K_{ij} \cdot D_{ij} - R \cdot D_{ij}^{-7}$$

where i and j are *situated modules*; F_{ij}, D_{ij}, and K_{ij} are the force, distance, and spring constant between i and j, *respectively*; and R is the constant for the force of repulsion. The spring constants are retrieved from the execution history. The position of each *situated module* is iteratively calculated until all positions converge.

The *Episode Editor* also helps our visualization of the recent execution history of *situated modules* (Fig. 7: upper half of the screen). The recently executed *situated modules* are connected and colored blue. The sizes are small because the modules were executed in the past. The current *situated module* is large and colored red. The candidates of the next execution are also large and colored green. Other *situated modules* are displayed very small.

Searching *Episode Rules*

The third function of the *Episode Editor* is to search implemented *episode rules*. There are three types of search: current search, recent history search, and direct search. In the current search (Fig.7), the *Episode Editor* shows the current status (upper half of the screen) and the *episode rules* (bottom half) match with the current status and history. In the history search, a user can specify a point in the history to search the *episode rules* that match a specified time. In the direct search, a user can select *situated modules* from the main screen to specify the situation (such as "<TURN=success><GREET=greeted >") to search the *episode rules*.

3. Implementation and Experiment

In this section, we intend to demonstrate the performance of the *episode rule* and *Episode Editor* by introducing our implementation and a simple experiment.

3.1. An Interactive Humanoid Robot "Robovie"

We developed a robot named "Robovie," shown in Fig. 8. This robot, which has a human-like appearance, is designed for communication with humans. In order not to alarm humans, we decided on a size of 120 cm. The diameter is 40 cm. The robot has two arms (4*2 DOF), a head (3 DOF), two eyes (2*2 DOF for gaze control), and a mobile platform (2 driving wheels and 1 free wheel).

The robot has various sensors: 16 skin sensors covering the major parts of the robot, 10 tactile sensors around the mobile platform, an omnidirectional vision sensor, 2 microphones to listen to human, and 24 ultrasonic sensors. We developed sensitive skin sensors using pressure sensitive conductivity rubber. By using the actuators and sensors, the robot can generate enough behaviors for communication with humans. It has a Pentium III PC on board for processing sensory data and generating behaviors.

3.2. Implementation

Based on the architecture previously described, we have implemented 102 *situated modules* (shown in Fig. 6) in the robot. It autonomously exhibits friendly behaviors, such as a shaking hands (Fig. 10-left), hugging (Fig. 10-center), playing paper-scissors-rock (Fig. 10-right), and short conversations. It speaks more than 350 sentences and recognizes over 50 words in Japanese. It sometimes expresses idling behaviors such as "scratching its head" and "folding its arms." It sometimes performs its daily work, which is patrolling around its environment.

We explain some of these behaviors in detail to show how the indication-recognition pair of *situated modules* realizes an active interaction. By executing the *situated module* "JANKEN." the robot says "Let's play paper-scissors-rock " and it starts stretching out its hand at the indication part. If the human reacts to the robot's action, the human will stretch out his/her hand near the robot's hand. Then, the robot recognizes the human's reaction by just gazing at its hands. As another example, by the *situated module* "FROM," the robot asks the humans where they are from. Then, it just recognizes place names.

We have also implemented 884 *episode rules*, on the basis of the idea that until humans interact with the robot, it goes around its environment and exhibits idling behaviors; once a human starts interacting with it, it initiates behaviors until the human stops reacting to them.

3.3. Experiment

We performed an experiment to verify the effects of the visualization support function of the *Episode Editor*. By using the implementation, the robot autonomously interacts with humans. We use two subjects (university students). Each of them interacted with the robot for ten minutes.

Result

Figure 9 indicates the result. There are two findings from the visually displayed result:

1. Meta-structure

When the robot interacts with humans, it has two states: playing with humans and showing idling behaviors. These were displayed visually on the screen. There are two clusters in both screens of Fig. 9. The upper-right clusters are the *situated modules* for showing idling behaviors, and the center-to-bottom-left clusters are the playing behaviors. Thus, the *Episode Editor* shows the meta-structure of the *situated modules*. Rarely executed modules are separated in the upper-right and bottom-left corner.

2. Difference between subjects

Regarding the screen for subject 2, the center-to-bottom-left cluster nearly separates into two (center and bottom-left). This separation does not appear on the screen the subject 1. To analyze in detail, we found that this occurred because subject 1 played with the robot longer than subject 2. The center cluster of the screen for subject 2 consists of the *situated modules* such as greet and handshake that the robot exhibits at initial playing behaviors. The robot exhibits such behaviors first, and if the human continues to react to such friendly behaviors, the robot continues to exhibit friendly behaviors. Subject 2 did not respond so eagerly to the friendly behaviors, therefore, the interaction did not continue so long for subject 2. Thus, the initial playing behaviors are clustered and displayed separately from other playing behaviors.

Discussion

The results indicate that the *Episode Editor* can help robot developers visualize the meta-structure of human interactions and individual variations within those interactions. We believe such intuitive visualization is important and necessary for our constructive approach to implement a large number of behaviors and their relationships. In addition, we are planning an experiment with many subjects, in which we will retrieve *episode rules* from the experiences of the interaction between the robot and the subjects who interact well with the robot.

4. Conclusion

We proposed a robot architecture that consists of *situated modules* and *episode rules* for interaction-oriented robots. The architecture had the following features: situated recognition to understand complex human behaviors, active interaction to make humans adaptively respond to the robot's actions, and *episode rules* to realize communication for a consistent context. Because a *situated module* is designed to work under a particular limited situation, developers can easily implement a large number of situated modules. Then, the *Episode Editor* helps to develop the complex relationships among many *situated modules* as *episode rules*. As the result of this experiment, the *Episode Editor* has enough visualization support ability for humans to understand the relationships and execution of interactive behaviors. We consider that a development support tool such as this is indispensable for realizing interaction-oriented robots that have a large number of behaviors.

Acknowledgement

We express our gratitude to Hitoshi Miyauchi (Mitsubishi heavy Industries, Ltd.) for the implementation of the *Episode Editor*. This research was supported in part by the Telecommunications Advancement Organization of Japan.

References

[1] C. Breazeal and B. Scassellat, "How to build robots that make friends and influence people," *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 1999.

[2] G. Cheng and Y. Kuniyoshi, "Complex continuous meaningful humanoid: A Multi Sensory-Cue Based Approach," *Proc. IEEE Int. Conf. on Robotics and Automation*, 2000.

[3] T. Kanda, H. Ishiguro, T. Ono, M. Imai, and R. Nakatsu, "Development and evaluation of an interactive humanoid robot Robovie," *Proc. IEEE Int. Conf. on Robotics and Automation*, 2002.

[4] H. Ishiguro, T. Ono, M. Imai, T. Maeda, T. Kanda, R. Nakatsu, "Robovie: an interactive humanoid robot," *Int. J. Industrial Robotics*, Vol. 28, No. 6, pp.498-503, 2001.

[5] S. J. Cowley, and K. F. MacDorman, Simulating conversations: The communion game. AI & Society, 9:116-137, 1995.

[6] T. Ono, M. Imai, and R. Nakatsu, "Reading a robot's mind: A model of utterance understanding based on the theory of mind mechanism," *Advanced Robotics*, Vol. 14, No. 4, 2000.

[7] M. P. Georgeff, A. L. Lansky, and M. J. Schoppers, "Reasoning and planning in dynamic domains: An experiment with a mobile Robot," *SRI International Technical Note 380*, 1987.

[8] T. Ishida, Y. Sasaki, K. Nakata, and Y. Fukuhara, "A meta-level control architecture for production systems," *IEEE Trans. on Knowledge and Data Engineering*, Vol.7, No.1, pp. 44--52, 1995.

[9] R. A. Brooks, "A robust layered control system for a Mobile Robot," *IEEE J. of Robotics and Automation*, 1986.