A Robot Programming Model for Mediating Between Familiarity-Oriented Behaviors and Environment-Oriented Behaviors

Akihiro KOBAYASHI, Izuru KUME, Atsushi UENO, Yasuyuki KONO, Masatsugu KIDODE
Graduate School of Information Science, Nara Institute of Science and Technology
Takayama-cho 8916-5, Ikoma-shi, Nara, 630-0192 Japan

ABSTRACT

In this paper we propose a novel programming model for a personal robot that has an original behavior and is provided with a new one by various working environments. How to coordinate robot's devices is an inevitable problem in robot programming. Normally the coordination is the responsibility of applications. In our case, however, we can't expect any coordination by the applications for the original and the new behaviors, because they are developed independently. We introduce a middleware that mediates accesses to robot devices. We point out basic concepts to qualify mediation, and propose a new robot programming model for those applications that promote mediation quality and cope with undesirable mediation results, it any.

Keywords: Robot Middleware, Personal Robot, Mediation of Multi-Application, Human-Impression of Robot Actions

1 INTRODUCTION

In this paper, we propose a middleware and a programming model to give personal robots the ability to supply local specific services in each working environment while keeping their original familiarity. Recent researches provide two viewpoints of the usage of autonomous mobile robots. On one hand, a robots is thought as a mobile and intelligent interface to information systems [2, 17, 11]. On the other hand, robot owners expect their robot to behave as a familiar amusement creature like a pet. Human impression on their appearance and motion is one of the most important issues [7, 8].

In near future, a personal robot which accompanies its owner should gain the role of interface to the information system in a working environment as long as they stay there. This ability gives benefits to both the owner of the robot and the service provider of the environment. ¿From the owner's viewpoint, he'd like to access the system through his familiar robot. The information system could use the personal robot as its physical interface for the visitor. Otherwise the service provider should prepare enough number of robots to provide these services. The benefits are summarized in the following.

- Owners
  - Robot owners are provided with services adapted to the current environment.
  - Robot owners can access the services through his familiar interface.
  - Adopted behaviors shows a change that prevents owners from getting bored with mannerism.
  - Robots in an environment would collaboratively work without any collision.
- Service Providers
  - It is not necessary to stock many robots for providing service; e.g., guiding visitors.
  - The robot would follow TPO rules without its efforts.

A robot owner feels familiar with original behaviors of his robot. On the other hand, new behaviors introduced by a local information system might make his robot behave distantly to its owner, though they provide useful service. Both kinds of the behaviors have a requirement about the robot's physical states so that the robot can accomplish them. Each behavior's execution normally changes the physical states. From the viewpoint of an application, the robot's state might be changed during its execution by its counterpart application. If the changed state doesn't satisfy the requirement of the application's behavior, possibly it can't be executed correctly. We call such a situation an *interference* by the counterpart's behavior. We say also that the counterpart's behavior *interferes* with the application's behavior.

Although both of the behaviors had better to be executed in parallel, the robot must prioritize one to another if it detects interference. In the following we

will see an example of interference between two applications in a robot, one application implements a pet-like motions while another navigation service in a building. We will also see how the robot prioritize the latter's behavior to the former's in order to avoid interference.

One application implements a pet-like behavior in a robot. For example, the robot occasionally turns around its owner with its pretty gestures and moves toward the owner as if a pet runs toward its beloved owner. Then it keeps around its owner for a short time. The program uses a way of visual feedback based on a human recognition technique. This familiar behavior takes place anywhere and anytime in order to entertain its owner. Another application implements a service to guide visitors to their destination. This program uses a navigation technique based on a topological navigation technique. The guide service shows a user friendly behavior that a guide robot prompts a visitor to which directions he should go at a corner in the building.

When a visitor and his robot enter the building, the robot begins to guide its owner while it occasionally shows the pet-like behavior at the same time. They can cope with the parallel execution of the both applications by adjusting the movement vector and keeps the owner in its view. (Fig. 1 a) However, when they come to a corner and the navigation behavior prompts the owner (Fig. 1 b), the pet-like behavior should be stopped for two reasons because the robot should prevents the owner from being confused. There is another reason to stop the pet-like behavior that the robot should watch the owner not to go in wrong direction. Hence keeping him in its view is critical at this moment. (Fig. 1 c) The robot should restart the pet-like behavior as soon as after they leave the corner in order to entertain the owner.

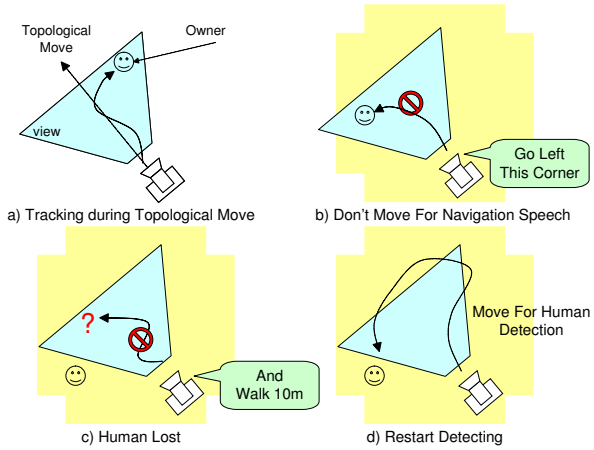

Figure 1: Example of Simultaneous Execution

Notice that the two applications in the above example are not implement to cooperate each other. The robot, however, could act correctly because it has a mechanism to avoid interference, which is the theme of our research. We call the mechanism *mediation*. This paper explains the concepts of mediation and a new programming model to use the mechanism. It is new theme in robotics to integrate software components both of which never know how their counterpart manages their common resource, i.e., the physical devices in the robot. We introduce a new architecture that separates the management responsibility from applications. A middleware, called a mediator, has the responsibility instead. Hence, specification of a mediator and its programming model are key issues of our research. We categorize robot application and examine their natures and requirements in general. Then we can introduce important factors for a mediator to decide its mediation. We propose a new robot programming model, including programming guidelines and basic concepts to realize the higher level of mediation.

2 MEDIATION

Traditional Robot-Programing

A new framework of mediation is needed for a robot to execute multi-applications in parallel which are developed independently, because robot programmers traditionally have developed whole applications in the robot on the condition that unknown applications will never execute. It is necessary to control robot's motion consistently, because a robot has accesses to the real world. However, it's too difficult to keep the robot secure while it executes independent applications in parallel.

There is always a gap between a real world environment and its representation in a robot gained through its error-prone sensors. Robot programmers adopt the reactive programming method to narrow that gap by repeatedly accomplish short-term goals like obstacles avoidance, because they believe the latest sensor-input more than the past information. On the other hand, a robot application has heavy calculations or long-term goals such as image recognition or global path planning in general.

Brooks proposed the Subsumption Architecture [1] to cope with the both goals. Subsumption Architecture model represents a process as parallel data flows from sensors input to output by actuators. The model introduces layers of system modules according to the various levels of goals. Long-term goals are implemented in upper layers and short-term goals in lower layers. Interventions from upper layers to lower layers are allowed. (Fig.2)

Designing under Subsumption Architecture, a programmer needs to understand details of lower layers

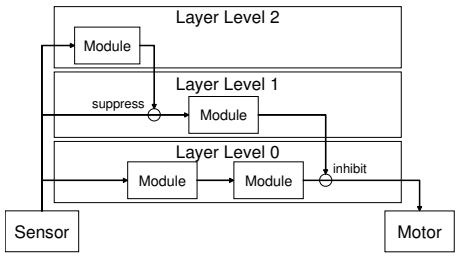

Figure 2: Subsumption Architecture

before designing upper layer.

Mediation Framework

In order to cope with dynamic coordination, we propose a new process model based on *mediation* instead of coordination. We assume the middleware on which all applications run. The middleware provides applications with basic functions for a mobile robot and mediates any access to devices from its applications. Subsumption Architecture coordinates applications with respect to devices accesses. The proposed middleware is an online interpreter that can block device accesses and can schedule threads dynamically using only the behaviors and additional information. We call the middleware *mediator*. (Fig.3)

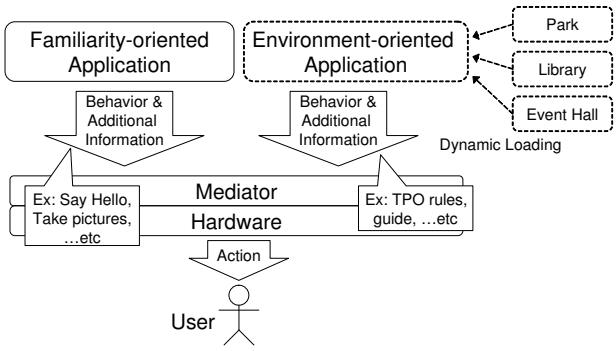

Figure 3: Mediation between Each Application

As Fig.3 shows, a personal robot has its original application to represent familiar actions. We call such an application a *familiarity-oriented application*. On the other hand, the robot loads dynamically a new application to perform the information service in the current environment. We call such an application an *environment-oriented application*.[1] In this paper, we name a request from an application as *behavior*. Each behavior often interferes with each other.

[1]Usually applications and actions of robots are traditionally called as behavior.

Applications with Different Features

The mediator should take care of the difference between the both applications in order to solve the interference. A familiarity-oriented application is designed with robot hardware usually by a same designer. It provides familiar actions which make user comfortable. Therefore, it must include many operations to drive motors in order to achieve delicate motions. The mediator should keep minute features of these motions.

On the other hand, developers of an environment-oriented application tend to write the code with little information about robots' physical feature. An application associated with an environment controls unknown robots witch will come into the environment. It is impossible for them to predict details of robots accompanied with their owner. The programmers must write their codes to command a robot in an abstract fashion such as navigation from a place to another. The applications need to accomplish useful services and follow TPO rules more seriously than to duplicate details of actions. The mediator should support basic functions for a mobile robot, and they should be robust enough.

Requirement to Mediation

The mediated actions should fulfill specific requirements of users by estimating the trade-off among the requirements mentioned in the subsequent section. The mediator evaluates the quality of performance of robot's behavior from three aspects, i.e., *reliability*, *familiarity*, and *efficiency*. Reliability guarantees to accomplish services, and to follow TPO rules in the environment. It is one of the "must" of a robot's performance. For example, the behavior generated of another application would prevent correct speech recognition, or break the context of interaction. The mediator should solve those problems. Reliability also guarantees reactive actions, so that they need for a robot to deal with the real world environment.

The familiarity means the degree of how the owner is impressed by its actions. The efficiency is measured by the time for a robot to spend in its actions. The user hopes for his robots to take familiar actions and to accomplish his tasks in a matter of minutes. We can find trade-offs among familiarity and efficiency. For example, the familiarity of a robot is high, if it often performs its familiarity-oriented behavior together with environment-oriented behaviors. On the other hand, it also lowers the efficiency in terms of the performance of the given task.

3 POLICY OF MEDIATION

Familiarity
Mediation gives the way of selection in the trade-offs. We adopt a policy that the familiarity should be preferred to the efficiency as long as it keeps certain amount of the reliability, because a personal robot should be familiar with its user. We suggest some hypotheses on the sources of familiarity, so that the mediator can objectively measure familiarity that is the user's impression to the robot. We make an assumption that the robot satisfies the familiarity the best, when the robot performs the familiarity-oriented behaviors faithfully to its original ones. However the mediator would suspend, divide or refuse some behaviors depending on the situation that the robot faces. ¿From a practical viewpoint, it is reasonable to decide the guidelines for a robot to perform with better Familiarity. We assume the following factors increase Familiarity aspect;

- Similar motions in the familiarity-oriented behavior and its total performance time
- Minimising the suspended time of the familiarity-oriented behaviors
- Seamless switching between the familiarity-oriented behavior and the environment-oriented behavior
- Quick responding to the user as possible

Mediation Layers
One of the following four layers of mediations is selected, whenever the mediator gets requests from one of the applications while executing requests of the other. Higher layer is more familiar, because the suspended time of the familiarity-oriented behaviors becomes smaller when the mediator selects higher layer. More programmers' efforts are required to realize more familiar mediation in general. The mediator tends to select higher one, as far as the mediator keeps the reliability enough.

- **Semantic Execution:** Both behaviors requested by the applications at the same time run efficiently. The mediator understands the semantics of the behaviors and generates new actions to fulfill their features.
- **Concurrent Execution:** The both behaviors run simultaneously. The mediator fuses commanded motions and cut a part of behaviors to exclude the part of behaviors which conflict with requests from the other application.
- **Time Sharing Execution:** The each behavior runs exclusively at one time. The mediator makes a time schedule for exclusive execution. The mediator interrupts running behaviors at a "safe" point, and resumes the suspended behaviors.
- **Sequential Execution:** Each behavior runs using batch processing. The mediator injects interruption, and follows the sequence to execute details of actions faithfully.

4 PROGRAMMING MODEL

Programming Guidelines
As we explained in section 2, environment-oriented behaviors and familiarity-oriented behaviors possibly interfere in accesses to devices. As we explained in section 3, we have an assumption that it is happy for the both sides of the behaviors if neither one occupy a robot's devices for long. For the sake of refined mediation, we propose three programming guidelines. First, a programming code must specify in their code the requirement to access the devices exclusively in order to require reliability.

Second, the code should permit its counterpart to access devices as long as its own reliability is guaranteed. We call a programming code written with this principle *a reconcilable code*. Applications written in a more reconcilable code enable higher mediation, because the mediation can allot each side to the devices one after another in a short time. Third, programmers should expect and cope with the situation that its request will be refused or interrupted by its counterpart. The programmers cannot avoid such a case because mediation might prioritize the counterpart at runtime.

Reference Model of Robot's State
As Application often refers and changes physical states of a robot in the real world environment. Environment-oriented application programmers must write their code without physical details of the robot. Therefore, they need one common reference model that abstracts physical states of robots equipped with a variety of physical devices. The mediator in a robot must understand the current state of the robot from its sensor-inputs, and must make motor-outputs to change the current state into the next state required by the application. Therefore, the mediator is responsible for the bi-directional translation between abstract states in application and physical states.

Recent fruitful research results in mobile robots [13] show the maturity of mobile techniques, which help our work to construct the reference model with respect to mobility. We assume the existence of a map of the working area, which we restrict to inside of building of office or public spaces. We have also assumed a universal way of tracking that is less dependent on robot's hardware configuration and environment. As

for the latter means, we believe that combination of topological navigation [9, 10] and tracking technique using ceiling image [15] satisfies our requirements.

Although our work is now in progress, we have already pointed out several basic concepts of the reference model with respect to mobility as follows:

- robot's coordinate position in a map
- topological position in a map
- prohibition zones
- robot's direction
- obstacles
- robot's velocity
- robot's attitude
- device occupation

All of them are crucial especially for programmers of environment-oriented applications. Coordinate position is a "exact position" on a map provided by environment-oriented application programmers.[2] In general, robot navigation often accepts a range of positioning error and topological positions are used for navigation programs instead of coordinate positions. Prohibition zones are specified in either of style positions. They include those places where guests and their robots should not enter as well as blind spots for position recognition by robots. The direction means absolute bearing in a map. Obstacles mean those objects, which don't exist in a map but detected by the sensors of a robot. Attitudes mean whether the robot is ready to move. Some robots have a complex body cannot move at any time. Device occupations means that one of the application needs to access the device exclusively.

Embodiment-Based Exception Handling

As we explained in section 2, a programmer should describe additional informations for a better mediation. For that purpose, the programmer should clear (1) where he needs to protect the robot's action in his code, (2) what he needs to protect, (3) how much he needs, and (4) what he needs to do when the protection is broken. We introduce a programming style, called *embodiment-based exception handling* to express those requests. An exception handling consists of four factors programming constructs, *block, definition, seriousness, handler.*(Fig.4)

A *block* is defined by an area of an application codes. The programmer describes it using curly brace, and so on. The robot cannot accomplish some behaviors, because of the actions of the counterpart application. The counterpart application may break the critical control, cause the user's confusion, or interfere with sensing. The programmer should signify the *block* where the behavior needs the particular state of the robot in its codes.

A *definition* is defined by a state of the robot or *refusal exception*. A programmer describes the state of robot using the reference model. The *refusal exception* is defined as a special state by the mediator. When the robot becomes in the exception state in the block, the mediator throws the exception defined, and move the control to the handler associated with the exception. For example, if the programmer wants the robot to wait some guests in entrance hall, the robot must keep in the entrance hall during waiting guests. He should define the state that the robot is away from the entrance hall as the definition of an exception, and should define the area of his codes waiting guests as the block of the exception. The *refusal exception* is thrown when the requested command is refused because of the conflict with the counterpart application. The programmer can describe the handler to deal with the refusal because of unexpected causes.

Seriousness defines whether the application can compromise or requests the refusal of the counterpart. If an exception is defined as serious, the mediator assigns high priority to protect from the exception with restricting requests by the counterpart. A handler defines what the application should do, when the exception signal are thrown. There is one handler associated with one definition of exception. The programmer should describe exception handling for the robot to take correct actions, i.e., breaking of loops, initialization of local parameter, memory allocation, and so on. Even if the programmer specifies an exception as serious, he should define the handler of the exception. The exception may arise, when the counterpart interferes with the execution in the block. Programmers should avoid unnecessary serious specifications, because they interfere with those block witch are serious.

5 REALIZATION

Mechanism

We explain the internal mediation mechanism. The mediator intercepts all requests from applications to access the devices. It abstracts sensor-inputs and actuator-outputs as a state in the reference model in section 4. Application requests the mediator to enter a block. If one application is in a block of a serious exception and its counterpart is not, device access requests by the latter are ignored by the mediator when any interference is detected. If both applications are in their block and the both definitions have any inconsistency, the mediator should select one application and

[2]We assume the existence of a standard format of maps

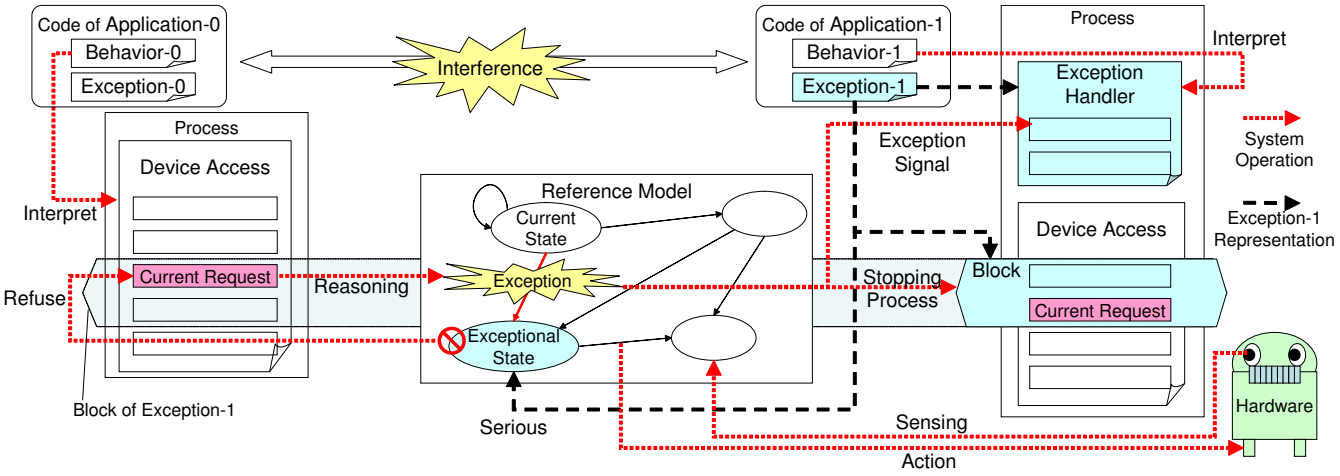

Figure 4: Model of Exception

throws an exception to the counterpart. The middleware makes a selection based on the seriousness of the blocks and its mediation policy explained in section 3. The exact selection mechanism depends on the mediator's implementation.

There are two types of exceptions according to the timing to detect interference. A *refusal* exception notifies refusal of an access request. A *state* exception notifies an application that its counterpart accessed a device and set the robot in an undesirable state. Applications thrown an exception by the mediator exit their block and move their control flow to the handler corresponding to the exception, if any.

Fig. 4 shows a snapshot of a mediation process. The mediator has just detected that one application (denoted as "application-1") requests a device access (denoted as "current request") which causes new state. Because its counterpart (denoted as application-0) specifies the state as a serious exceptional state, the mediator refuses the request and throws a refusal exception to application-1. Upon the refusal, the control flow of the application-1 move to the corresponding handler (denoted as Exception Handler).

Example Mediation

Now we go back to the example in section 1. In order for the robot to act as described in the example, application programmers must implement appropriate embodiment-based exception handling code as well as the behaviors. We assume that the programmers in each side obey the programming guidelines explained in section 4. The environment-oriented application programmers believe that its counterpart's behavior should be depressed during the behavior despite the second rule in the guidelines. The familiarity-oriented application programmers, on the other hand, are tolerant of interference by their counterpart, that is, any environment-oriented application during the pet-like behavior.

The environment-oriented application programmers specify the physical state required during the prompting behavior. They think that the robot should stop near the corner and track its owner in order to watch him. They specify their requirement by defining two embodiment-based exceptions, immobility and human tracking. The programmers can describe the definitions in terms of the reference model explained in section 4. [3] They specify a block to include the code for the prompting behavior. The block is specified as serious in order to reflect the programmers' above consideration with respect to interference. Then they implement a code to handle the rise of the exceptional state. The handler codes were not executed in this example and we thus omit their explanation. As for the pet-like behavior of the familiarity-oriented application, its seriousness is specified not so serious because of the above consideration by its programmers. Therefore, the application tends to catch a refusal exception during the behavioral execution. The handler simply repeats the whole execution from its start as soon as possible.

Of course the mediation result between the two behaviors depends on the mediator's implementation. However, we can expect that many mediators' respects the seriousness assigned to the block of embodiment-based exception handling code. The mediator in this case also prioritizes the environment-oriented behavior to its counterpart because of the assigned seriousness, and thus we gain the result in section 1.

[3] Notice that environment-oriented applications are written for anonymous robots which will accompany the visitors to the environment.

6 DISCUSSION

Realization

The mediator should get over the variety of robots and environments. In that point, it isn't difficult to implement the mobility of robots into a middleware. There are two reasons for it. First, it is necessary for programmers to express robot's actions, that the outputs of robots are defined as general descriptions. It is easy for robot navigation, because the outputs of robots can be expressed with 2-dimensional vectors. Second, robust recognitions of environments should be implemented into the mediator to enable every robot to use supported commands in everywhere. The functions of robust localization will be developed in the near future, under the favor of past researches [13].

Not only robot navigation, but also human-robot interaction should be supported by the mediator, to make robots more familiar. To realize it, the general model of human-robot interaction should be made, and the robust recognition about human is needed. Those problems are difficult, but we have a chance of robust communication on the whole, if the improved turn-taking rules effectively use cleverness of human.

Development Method

We didn't address performance issues in our programming model. In general, performance often affects robot programs. For example, navigation program should arrange robot's movement rate with the frame rate of its image recognition. In addition, to such general cases performance issues should be introduced into our programming model. Interference in device accesses possibly introduces extra motions of physical device which were not intended in original program codes. The interference might hold the essence of the original motion but affects its physical efficiency. It is a rational intention to assure the motion's good performance by preventing its counterpart's interference. Hence, the description of its mediation policy should include performance issues. Performance assurance may be requested by application programmers. For this purpose we need a means to define exceptions using performance issues.

How to describe performance issues is an open problem at this stage. We expect that they only appear in exceptions. Apart from implementation, how to design and check software with performance specification is also important. For this problem, we will find a method in a similar fashion to real-time software design and specification methods [3, 14]. How to model mediation policies from the view point of performance is an important problem, too. One of the key concepts of the policies is matching between requirement and supply of robot devices. We believe we can describe it in terms of balancing problem between demands and capacity of computer resources. It suggests that we can apply quantitative approaches such as the one by Graupner and et al. [5], for example.

At this stage, we imagine the notation of embodiment-based exception handling in a similar fashion to Java's exceptional handling notation [4]. We select this notation style because we specify a block, the duration of a mediation request, in terms of the lines in a programming code. It is familiar to many programmers who are used to "try and catch" exceptional handling style. However, there is a possibility that application programmers need another specification of the duration that doesn't correspond to blocks of lines. Such cases will arise, for example, if they want to request mediation while a robot navigates in a particular area. We believe aspect-oriented programming style [16, 6] suites such needs. We will need to extend the concept of "pointcut" so that we can specify various kinds of duration in which mediation requests should be applied. This programming style will separate embodiment-based exception handling as an "aspect" from main control flows, and thus decrease the efforts of software maintenance.

7 RELATED WORK

There are little attempts to give robots the ability to execute parallel multi-applications that are developed independently.

We try to measure human impression to decide the policy of mediation. Imai and Kanda [7, 8] studied about human-robot interactions at the view point of human impression. Their research aims to discover evaluation methods of human-robot interaction (both subjective and objective measurements), analyze social relationships between humans and robots, in order to introduce robots into real human society.

Our reference model of mobile robots should apply to various hardwares. Resent researches of middleware for robots will help us. ORiN [12] aims to standardize the specifications of interface and data and negotiation for robot controllers to communicate with various applications. It enables heterogeneous robots to communicate with each other, separates applications from robot controllers.

As for performance issues, real-time UML [3] provides a method to express real-time design model of robots' behaviors. It augments UML and introduces performance specification into diagrams. We can extend the specification with more aspects in order to evaluate their quality.

8 CONCLUSION

In this paper, we propose a middleware and a programming model to give personal robots the ability to supply local specific services in each environment

while keeping their original familiarity. We aim at familiarity from them, and categorize mediation method along the policy of mediation for personal robots. We suggest the programming tool to enable high level mediation. The programmer defines exceptions and their handler to keep the desirable state of the robot without excessive occupation. The capability of these commands depends of description of the robot model. We analyze the mobility of robots, but more efforts to generalize the robot's embodiment are needed to mediate the human-robot communication in this system.

References

[1] R. A. Brooks. A Robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation, RA-2*, 2(1):14–23, March 1986.

[2] J. Buhmann, W. Burgard, A. Cremers, T. H. D. Fox, F. Schneider, J. Strikos, and S. Thrun. The Mobile Robot Rhino. *AI Magazin*, 16(1):31–38, 1995.

[3] B. P. Douglass. *Real-Time UML*. Addison-Wesley, 1998.

[4] J. Gosling, B. Joy, and J. S. Guy L. *The Java Language Specification*. Addison-Wesley, 1996.

[5] S. Graupner, V. Kotov, and H. Trinks. A Framework for Analyzing and Organizing Complex Systems. In *Engineering of Complex Computer Systems*, pages 155–165. IEEE, 2001.

[6] Gregor Kiczales and Erik Hilsdale and Jim Hugunin and Mik Kersten and Jeffrey Palm and William G. Griswold. An overview of AspectJ. In *ECOOP*, 2001.

[7] M. Imai, T. Kanda, T. Ono, H. Ishiguro, and K. Mase. Robot Mediated Round Table: Analysis of the Effect of Robot's Gaze. In *Proc of The 11th International Workshop on Robot and Human Communication (RO-MAN)*, pages 411–416. IEEE, September 2002.

[8] T. Kanda. *A Constructive Approach for Communication Robots*. PhD thesis, Kyoto University, 2003.

[9] B. J. Kuipers and Y.-T. Byun. A Robust, Qualitative Approach to a Spatial Learning Mobile Robot. *SPIE Sensor Fusion : Spatial Reasoning and Scene Interpretation*, 1003:366–375, 1988.

[10] M. J. Mataric. Integration of Representation Into Goal-Driven Behavior-Based Robots. *IEEE Transaction on Robotics and Automation*, 8(3):304–312, 1992.

[11] T. Matsui, H. Asoh, J. Fry, Y. Motomura, F. Asano, T. Kurita, I. Hara, and N. Otsu. Integrated Natural Spoken Dialoge System of Jijo-2 Mobile Robot for Office Services. In *Proc. of The 16th National Conference on Artificial Intelligence (AAAI-99)*, Florida, July 1999.

[12] M. Mizukawa, H. Matsuka, T. Koyama, T. Inukai, A. Noda, H. Tezuka, Y. Noguchi, and N. Otera. ORiN: Open Robot interface for the Network, The Standard Network Interface for Industrial Robots and its Applications. In *Proc. of International Symposium on Robotics (ISR)*, 2002.

[13] R. R. Murphy. *Introduction to AI Robotics*. MIT Press, November 2000.

[14] A. C. Shaw. *Real-Time Systems and Software*. John Wiley & Sons Inc, 2001.

[15] H. Tani, Y. Matsumoto, and T. Ogasawara. Indoor Navigation Based on Ceiling Images - Automatic Mosaic Method for Building Ceiling Maps-. In *Proc. The 19th Annual Conference of the Robotics Society of Japan*, pages 1013–1014, 2001.

[16] the AspectJ Team. *The AspectJTM Programming Guide*. Zerox Corporation., 2002.

[17] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. NINERVA: A Second-Generation Museum Tour-Guide Robot. In *Proc. of the International Conference on Robotics and Automation (ICRA)*, pages 1999–2005. IEEE, 1999.