Behavior description and control using behavior module for personal robot

Yukiko Hoshino
Sony Corporation
Life Dynamics Laboratory
Preparatory Office
6-7-35 Kitashinagawa,
Shinagawa-ku, Tokyo,
141-0001 Japan
Email: yukiko@pdp.crl.sony.co.jp

Tsuyoshi Takagi
Sony Corporation
Entertainment Robot Company
6-7-35 Kitashinagawa,
Shinagawa-ku, Tokyo,
141-0001 Japan
Email: takagi@pdp.crl.sony.co.jp

Ugo Di Profio
and Masahiro Fujita
Sony Corporation
Network CE Laboratory
6-7-35 Kitashinagawa,
Shinagawa-ku,Tokyo,
141-0001 Japan
Telephone: (+81) 3-5448-5901,
Fax: (+81) 3-5448-6833

***Abstract*— This paper describes a module-based behavior selection architecture for a personal robot intended for a real world environment. We adopt the Emotional GrOunded architecture for a basis, and define and describe a behavior module and an associated tree structure for controlling many behavior modules. Also we discuss the requirements and approach for controlling the behavior module tree. Through experimentation and implementation on QRIO SDR4X-II, we confirm the feasibility and design of the behavior selection system.**

I. Introduction

There are many new research efforts involving robots that live with human users in daily life in recent years. [1][2][3] To develop robots capable of sharing the life of a human for extended periods, it is indispensable that the robot not only perform useful tasks but also entertain people. To realize this concept, which we call "Robot Entertainment", the key issue is how to make the robot's behave like a living entity, and it is our belief that the solution lies in the realization of complex behavior.

We earlier proposed behavior modules and a preemption process for behavior modules in [4][5]. In this paper, we focus on integration of the behavior modules based on a tree structure and also on the behavior selection process. This behavior selection strategy, which goes beyond the earlier behavior selection architecture of AIBO, covers not only simple behaviors but also complex activities such as interaction with a human.

First we overview the basis of the behavior selection architecture and its requirements. Then we address the main issue, the tree-structured integration of behaviors.

II. EGO architecture

We proposed EGO architecture (Emotionally GrOunded Architecture) as a behavior control architecture for autonomous robots. [6] The main strategy for behavior selection of EGO architecture is based on ethological model[7]. The behavior control is based on homeostasis which let the robot regulate internal status within a certain range, and interpretation of external stimuli with corresponding behaviors. Figure 1 shows

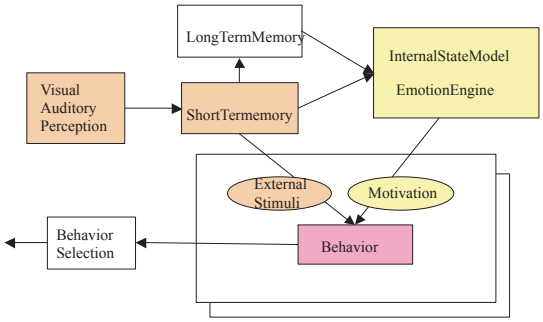

Fig. 1. Overview of EGO architecture

the outline of EGO architecture. The robot integrates visual, auditory and other sensing informations into the short term memory(STM) and uses those informations as external stimuli. On the other hand, it calculates motivation[7]from internal status and emotion. Using external stimuli, motivation and information from long term memory, the robot selects behaviors. In this paper, we focus on the bahavior selection part of the EGO architecture and describe the framework of expression, integration and control of complex behaviors, followed by description of our implementation on real robot system, QRIO SDR-4XII.

III. Requirements and approach for the control of various behavior

A robot should behave with sufficient complexity and in meaningful ways if it is to share the life of a human for a long period. The basic requirements for this kind of behavioral control include: (1)effective structure of the behavior, and (2)suitable coordination of the various behaviors. From an engineering viewpoint, it is also important for the developers of a robot's behaviors, (a) to provide a simple description, (b) employ a simple strategy for behavior design, and (c) make it easy to reuse behaviors.

To achieve these requirements, we propose an approach as follows: (1) Modularization of behavior, (2) Integration of behavior modules based on a tree structure, (3) Parallel

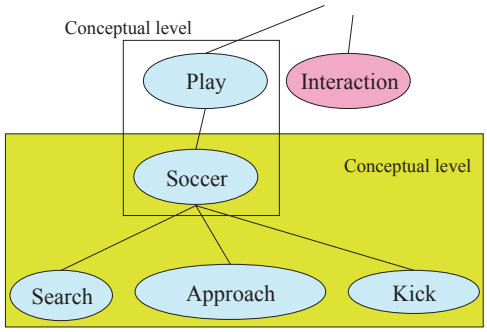

Fig. 2. Tree-structured behavior modules

evaluation and parallel execution of behavior modules, (4) Interruption continuation of execution , and (5) Behavioral selection based on behavior values (described later).

Modularization of behavior and integration of behavior modules based on a tree structure makes the design of a robot's behavior easy and simple. Parallel execution and interrupt-resume procedures permit an increase in the complexity of a robot's exhibited behavior. Moreover, the introduction of behavior values lets the behavior selection method be straightforward, as behaviors can only be selected that possess a suitable behavior value. Let us explain these points in detail.

1) **Tree structure for integration of behavior modules**
   (a) Modules are defined as a unit of behavior: Each behavior module has two functions; computing its suitability and the execution of actions.
   (b) Behavior modules are composed according to their meaning, and form a tree structure which has layers corresponding to conceptual levels.
   Figure 2 shows the tree structure of behavior modules. The figure's rectangle illustrates the conceptual level. The upper layer behavior module is termed the "parent", and its "children" are the lower layer modules.
   Due to the independence of behavior modules, both ease of their design and reuse can be achieved. Furthermore, the tree structure using a layered conceptual level helps in composing higher conceptual behavior with detailed behavior modules.
2) **Behavior value and behavior selection strategy**
   Within each behavior module, a behavior value that represents the suitability of the behavior is calculated. This value can be used to prioritize behavior, depending on the strategy chosen for controlling child modules. The high-level module in the tree structure only gathers the behavior values of its direct children so that the subtree size can be easily increased using the same behavior selection policy based on behavior values.
3) **Parallel execution**
   Each behavior module has certain keys for execution such as a robotic resource, and behavioral coordination using these keys enables the robot to exercise parallel execution.
4) **Interrupt-resume processing**
   In order to realize behavior interupt-resume, each behavior module maintains its own status, and performs suitable processing to change its status from active to waiting when a module which has higher priority than itself causes it to stop executing. That module can also resume execution and continue its behavior after being deemed a suitable process for reactivation.

As mentioned in the introduction, the details of the behavior module and interrupt-resume process were described in a previous paper [4]. In this paper, we focus on the integration of behavior modules based on a tree structure and on the means of behavior selection.

IV. Tree structure for the integration of behavior modules

The rationale for using a tree structure is: 1) Easy reconfiguration of behavior modules into different organizations, 2) a clear interface between the behavior modules and preservation of their independence, and 3) the abstraction of an entire subtree into a single behavior module.

In this section, we focus on the key issues surrounding the integration of behavior modules. These are six-fold; (A) Independence of each module, (B) Control at the conceptual level, (C) Common values and keys for execution, (D) Parallel evaluation and parallel execution, (E) Control strategy in each conceptual level, and (F) Shared information.

A. Module Independence

Ensuring independence helps in providing reusability of behavior modules. However, it requires a mechanism of behavior coordination to reside above the reused behavior modules. In our case, a parent coordinates its children behavior.

B. Control at the conceptual level

Because of the inherent complexity associated with large compositions of independent behavior modules, it is essential to have a suitable control strategy. Also, to support easy reconfiguration of the behavior module tree, it is important to minimize the amount of information to be shared by every behavior module.

Control of the whole behavioral tree is based on selection information available at each conceptual layer. The values considered are the priority for behavior selection, and an execution key for parallel execution. Control policies of the parent are based on only these two values.

The parent considers child modules as single behavior modules. There is no assumption that a child is a parent of another subtree. Moreover, each module can serve as both an executable behavior module and a parent module for coordination of other child modules.

C. Common values and keys for execution

To control behavior modules in each conceptual level, the information about their priority value and execution key are needed.

We adopt the term behavior value to represent an execution priority. This value is calculated from the motivation (depending on the robot's current internal status), and a merit value (depending on the current external stimuli), as follows:

$$Bv = \beta Mv + (1 - \beta)Rv \tag{1}$$

Mv is the motivation value which is calculated as a desire vector from the robot's internal state.

Rv is the releasing value which represents an expected merit value. It is calculated as follows:

$$Rv = \alpha \Delta S + (1 - \alpha)(S + \Delta S) \tag{2}$$

S is the satisfaction value derived from current internal status, and ΔS is the expected change of the satisfaction value as follows:

$$\Delta S = f(\Delta I) \tag{3}$$

$\Delta I$is the expected change of internal value which is based on current external stimuli.

Also, a behavior's required resources are referred to as an execution key. For example, one part of the robot's body can be used by only one module at a time. The Head resource cannot be used by two behavior modules simultaneously. The interpretation of language can be treated similarly, where this constitutes a cognitive resource. These keys form limitations for each behavior module when considered in terms of parallel execution.

D. Parallel evaluation and parallel execution

The behavior selection sequence starts from the lower level of the behavior tree, gathering the necessary information for selection towards the top. Execution is then assigned from the top level down to the lower ones.

The sequence is propagated from parent to child down to the leaf modules, which perform the evaluations of their behavior values and execution keys, and then transmit them to their parent, and so on.

After gathering all the information at the top layer, the parent module assigns an execution key to child modules at each conceptual layer, and propagates it sequentially from the top of the tree to the bottom. Behavior modules are then activated in parallel until a conflict of execution key is detected.

A conflict is detected by the parent module at each conceptual layer. In the case of a conflict, the module with higher priority is selected for execution. If the module with lower priority is already executing, it will be stopped first and then the higher priority module will be activated.

The execution status of each module is propagated from the child module to its parent module.

Thus, dynamic and parallel activation control of the behavior modules based on available execution keys is enforced at each moment.

E. Control strategy within each conceptual level

The evaluation and execution at each conceptual layer consists of:

- Gathering behavior information during an evaluation phase
- Distribution of key in execution phase
- Integration of behavior status

1) Gathering behavior information in the evaluation phase: Given a trigger for evaluation from its parent module, the child module calculates its behavior value and its key for execution. Gathering the information from all child modules along with those of the parent itself is indispensable for each conceptual level's control, especially for the distribution of keys in the execution phase.

The manner for gathering depends on the strategy for controlling child modules. For example, if the parent module wants to activate all child modules at the same time, all keys required by the children and the parent itself are needed for evaluation. On the other hand, if the parent module does not need to activate all child modules at the same time, a candidate list of execution keys is enough. Through this gathering and calculation, a behavior module can propagate its execution key to its parent module. The integration of priority value information is also needed.

This information collection process enables complete evaluation and execution in each concept level of the behavior tree, while also enabling proper distribution of keys from the top to the bottom layer.

2) Distribution of key in execution phase: Using information which is gathered and integrated as explained previously, a parent module distributes the execution key and manages the activation of its child behavior modules.

The key is distributed from parent to child and all available keys, except for the keys required by the parent itself, are distributed to child modules. The computed behavior value is used as the priority level for arbitrating key distribution.

After the distribution of keys, the parent module manages the activation status of its children according to the assigned execution keys. Child modules without a key should stop execution and release any keys they currently hold. Child modules not yet activated but possessing an execution key should start execution, when the assigned keys are released from the other modules which were previously holding those keys.

3) Integration of behavior status: After execution, a parent module must update its own behavior status and that of its children. In [4], we proposed adding an intermediate behavior status between run status and stop status . Figure 3 shows the set and organization of these behavior statuses. Two different stop statuses are used. The first stop has initialized information, while the other has preserved information for preemption and resumption of the module. To update the behavior status of its modules properly, the parent module has to integrate the information of its own behavior status based on the results of its execution, (e.g., the execution succeeded, failed, paused,

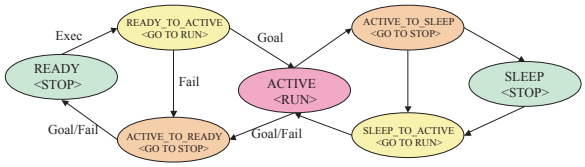

Fig. 3. Behavior status

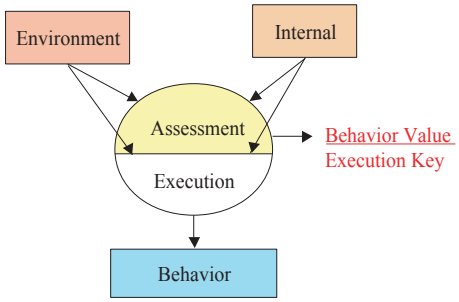

Fig. 4. Behavior Module

or continued), and the child modules' behavior status. This integration also depends on the strategy for controlling child modules.

At the same time, a parent module checks whether the entire subtree is able to continue execution or not. This also depends on the execution strategy employed. For example, if all child modules have to be activated at the same time, one child module in stop status will cause all other modules in the subtree to change from run to stop status.

F. Shared information

Previously, we discussed the information needed for the basic control of child modules. High-level coordination of behavior modules in the evaluation and execution phase also requires additional information, e.g. information about the target of a behavior.

For example, the target information of a soccer behavior is a ball. To coordinate an approaching behavior and a kicking behavior, information about the target ball should be shared, and the robot should kick the ball to which it approached. In a sense, information about the target of a behavior should be shared at a conceptual level within the behavioral tree.

V. Implementation of behavioral control

In this section, we present an implementation of the module-based behavior selection system as previously discussed in this paper and in [4].

A. Behavior module

Each behavior module has an evaluation part and an execution part. (Figure 4)

In the evaluation part, a module calculates the execution key that is needed for the execution phase, and also the behavior value which is regarded as the behavior's priority.

In the execution part, the actual behavior is described as state machines. Using a state machine structure, the module can select a suitable action for given internal values and external stimuli, and transition to the next state as needed. The state machine includes sending an action command, changing internal status, deciding control policy of child modules, and so on.

A behavioral module also maintains its behavior status. Basically there are four status states: stop status, preparation status for activation, activated status, and preparation status for stop. Also for the preemeption function, the stop status includes two different status states: an initialized status and preserved status. With the preserved status, the behavior module can be reactivated from its previous stop point after a brief interruption.

B. Shared information

The information shared within a subtree of behavioral modules is: (1) Environmental information of a target, (2) Associated information of a target.

An example of environmental information is target information that the robot actually senses, e.g. an object seen by the robot. An example of associated information is a key for associating one target with long-term memory data e.g. a face index, voice index, or word index.

Several behavior modules in the subtree share this information.

C. Control for tree structured behavior modules

As mentioned earlier, a required function for tree-structured behavior modules is the propagation of the timing of evaluation, execution, and behavioral status update. For suitable control, the integration of calculated values, distribution of execution keys, and integration of status update are indispensible. These important functions depend on the strategy of child activation chosen as discussed in the previous section.

We propose a child execution policy that allows designers to have a simple description of the behavior.

As examples of child execution policy, there are three typical policies: (1) Simultaneous execution policy, (2) Parallel execution policy, (3) Main child existence policy.

Using this method, calculations and operations which depend on the child execution policy are hidden, and all the designer has to do is declare which of the policies is used and the associated child modules involved.

1) Simultaneous execution policy: Parent module activates all child modules at the same time.

- Calculation of evaluated values
  This policy requires all modules to be activated and stay active simultaneously. Execution keys for the subtree are the collection of all execution keys for all behavioral modules. This subtree can be activated only when all of the execution keys are available from the parent module.
- Distribution of execution keys
  In the execution phase, the parent module checks whether enough execution keys are distributed by its own parent module, and then distributes the keys to itself and its child modules.

- Integration of behavior status update
  Even when there is only one stopped module, the parent module stops every child and itself. Accordingly, behavior statuses are changed to preparation status of stopping. After execution, the parent module checks the result of the execution of the child modules and itself, and decides whether to change the behavior status or not.

2) Parallel execution policy: Parent module activates as many child modules as possible until no conflict of execution key occurs. This is the most common arbitration policy.

- Calculation of evaluated values
  The execution key is the candidate list of child modules and itself, because the parent module wants to activate as many children as possible. Even though not all keys are distributed in the execution phase, the parent module distributes execution keys to the greatest extent possible.
- Distribution of execution keys
  The parent module distributes available keys to child modules, except the keys required by the parent itself, according to the priority of child modules. Modules with high priority value are assigned keys first. After key distribution, behavioral modules are stopped or activated according to execution keys.
- Integration of behavior status update
  The parent module keeps the subtree active while there is at least one behavior module active, including itself. Only when all behavior modules are stopped then the subtree itself stops.

3) Main child existence policy: The parent checks main child (distinguished) modules in updating behavior status.

- Calculation of evaluated values
  The calculation is the same as for the parallel execution policy.
- Distribution of execution keys
  The distribution is the same as for the parallel execution policy.
- Integration of behavior status update
  The result of the execution in the main child module is most important. If the main child module finishes execution, then the parent module stops all the other child modules. The result of the behavior tree depends on the result of the main child module.

VI. Experiment with QRIO SDR4X-II

Using the behavior selection implementation described in section V, behavior experiments were performed using QRIO SDR4X-II in a real environment.

A. System composition

Figure 5 shows an overview of QRIO SDR4X-II. Using this robotic system, we performed experiments for behavior selection and behavior execution. The right part of the figure shows the software components. This robot has inputs from visual, auditory, and tactile sensors. It has 38 DOF, LEDs on its eyes and ears, and a speaker for talking. The software components include a long-term memory and an internal state model, which manages internal values and emotional status. The short-term memory gathers all inputs and the resource manager distributes output commands.

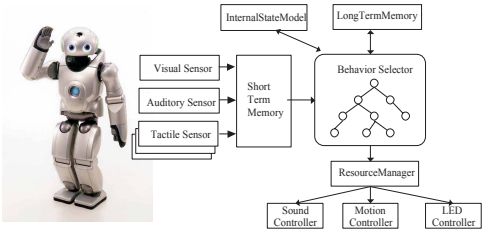

Fig. 5. Overview of robot system

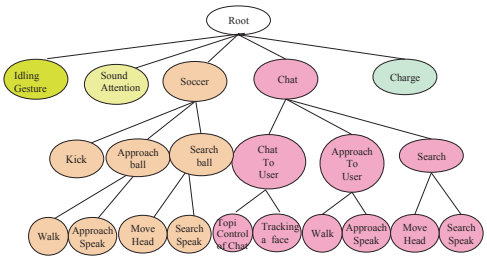

Fig. 6. Tree structure for behavior control experiment

B. Behavior control experiment with tree-structured behavior modules

Figure 6 shows an example of a tree of behavior modules. The behavior tree includes “Soccer”, “Charge”, and “Chat”. The main motivation of “Soccer” is vitality, the main motivation of “Charge” is hunger, and the main motivation of “Chat” is social interaction. There are other behaviors such as “Sound Attention”, where the robot turns towards the direction where user calls are detected, and “Idling gesture” where the robot performs idle gestures which has a constant behavior value. Figure 7 shows the whole experiment layout.

1) Layering of behavior in conceptual level: From a top level point of view, there are three kinds of behavior: “Soccer”, “Charge” and “Chat”. At the next layer, each subtree is defined: “Soccer” includes “searching a ball”, “approaching the ball” and “kicking the ball”; the “Charge” subtree includes asking to recharge its batteries; and the “Chat” subtree includes “searching a user”, “approaching to a user” and “chatting to the user”.

In addition, the “chatting to user” subtree includes the “tracking” a user’s face module and the “controlling topic” of chatting module. They are activated at the same time, according to the simultaneous execution policy.

2) Behavior selection:

- Using motivation: The behavior value of “Soccer” is increased according to the vitality internal state value. The soccer behavior tree is selected when the value of vitality is high enough. After searching, approaching, and kicking a ball, the robot is satisfied and vitality is decreased. The value of “Chat” changes based on

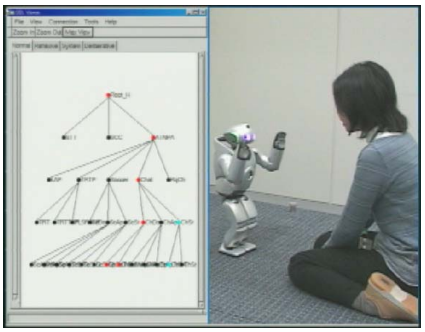

Fig. 7. Overview of an experiment scene

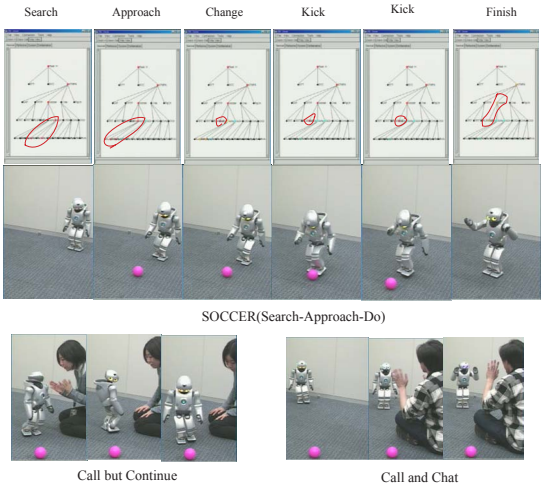

Fig. 8. Behavior selection experiment

the robot's need for social interaction, and decreases when the robot chats with a user. The behavior value of "Charge" increases as the battery charge level decreases. While the robot plays soccer and chats with a user, its battery will discharge and the motivation of the charge behavior will exceed those of both soccer and chat.

- Using motivation and external stimuli: First let's consider the behavioral changes inside one subtree, e.g. soccer. The Searching for a ball module and the Approaching module calculate their behavior values based on the existence of a ball and the distance between the target ball and the robot itself. When there is no ball in the robot's view, the value of the searching behavior is higher than the approaching a ball behavior. But when the robot finds a ball, the priority changes, that is, the value of approaching a ball becomes higher than that of searching. So the approaching behavior is selected. In the same way, the approaching a ball module and kicking a ball module compete based on the distance of the target ball. The upper part of figure 8 shows the robot's soccer behavior. We can now describe changes in activation between modules in different behavior subtrees. Suppose that someone calls the robot while the robot's approaching a ball. The robot stops and turns to the direction from where it was called, and finds a face. The behavior value of Chat will change based on the value of a user's face (who that person is) and its motivation for chatting. If the value of chatting is higher than the value of approaching the ball, the robot turns to the user and starts chatting. On the other hand, if the face is not so attractive for the robot, then the robot turns back to the ball, and continues its approaching behavior. The lower part of figure 8 shows the robot's different behavior depending on a user.

3) Parallel execution:

- Simultaneous execution: The Chat module consists of the simultaneous execution of tracking a user's face and controlling the topic of chatting. When the tracking module fails to find the target user, the result of the tracking module is "failure", and the behavior status of the tracking module changes to preparation for stopping. Then, according to the execution policy, at the same time the behavior status of chatting also changes to preparation for stopping and the robot stops chatting naturally.
- Parallel execution: The Approaching behavior consists of walking towards a ball, while speaking is an optional extension of the behavior. They are not needed to be activated simultaneously, so they are executed based on each one's timing.
- Main child existence policy: In the soccer behavior, the kicking a ball module is the main behavior module. So if the result of this main module is "success", then the result of the soccer subtree is also "success". But if the result of this main module is "failure", then the result of the soccer subtree is also "failure". On the other hand, the status of the other behavior modules in the subtree, e.g. approaching a ball module, has no influence on the status of the soccer subtree. If the result of the approaching a ball module is "failure", according to the change of behavior values, the searching for a ball behavior will be activated.

4) Shared information: In the soccer subtree, behavior modules share information about the target ball. With this information, the approaching module can know where the target ball is after the execution of searching module, and also the kicking module can kick the target ball which is approached by the approaching module.

Behavior modules in the Chat subtree also share information about the identified target user.

5) Preemption: Suppose that the robot is called by a user while approaching a ball. After turning to the user, the robot compares the behavior value of approaching with that of chatting. If the value of approaching is higher, then it can restart the approaching behavior towards the same target ball. That is because the information about the target is stored in the stop status of the approaching module.

Inside each behavior module, information about the state machine is stored. The module can restart from its previous

interrupted state. For example, a typical behavior of chatting is that the robot says "hello" to the user, does a self introduction and then say good-bye to the user. If a user calls the robot just after the robot has said "hello", then the robot will restart chatting from the self introduction.

VII. Discussion

First we address the issue of scaling of the behavior tree. Theoretically there is no limitation for the number of behavior modules in a behavior tree. The limitation is based on the computational power and the amount of calculation for each behavior. Currently, the result is that several hundreds of behavior modules are included in the behavior tree of QRIO SDR4X-II.

Next we discuss design criteria. It is important for the entertainment robot to facilitate the design of behavior, and that behavior modules can be written based on the designer's intention.

To design the robot's behavior easily, the modularity of each behavior module plays an important role. If the modules are independent relative to each other and can be connected anywhere within the tree, the designer can generate a large scale behavior tree using a high degree of reusablity while exploiting the strategy that the behavior tree can be made up of small subtrees which each represent independent functions.

That enables the designer to easily change the behavior tree and permits easy addition of new behavior. The designer does not have to create the entire tree from the beginning, but makes it expandable in a step-by-step process.

In addition, we prepared a basic policy for controlling the design of the tree, so that the designer can connect behavior modules easily. On the other hand, the way to write behavior modules is highly flexible, as it also enables the designer to write detailed control for the behavior tree if desired.

As mentioned above, the designer considers the system based on behavior modules and the way of connecting the behavior modules.

The next issue regards suitable behavior selection. As this behavior module architecture has high modularity, learning their evaluation functions is important. The behavior value is based on internal value and external stimuli. So it is required to learn suitable mappings between internal state values and the satisfaction value, between internal state values and the driving value, and between external stimuli and the expected change of satisfaction.

Another issue regarding learning of the behavior is learning a behavior as a connection of the behavior modules. Each behavior module has some actions, and the parent module gathers child modules as a subroutine of that behavior.

We intend to describe and discuss these learning issues in detail in another paper.

There is one more issue for behavior control. That is the chattering problem. If the behavior values of behavior modules are too close, a change of active behavior modules occurs too often. To avoid this, we adopt a lateral inhibition mechanism in our system. If one behavior module is active, then other modules in the same layer are inhibited based on the behavior value of the active module. This mechanism comes from our research on the emotional model [8] It provides for stability of the behavior, while also allowing for a change between active behavior modules if another module has a very high behavior value.

VIII. Conclusion

In this paper, we have described an architecture for the selection and activation of behavior modules. We have discussed requirements in a real world environment and presented our approach and an implementation of those ideas in a complex robotic system. We have also conducted experiments with the entertainment robot QRIO SDR4X-II and confirmed the suitability of the architecture. Finally, we showed examples of behavior trees and discussed control policies between parent and children modules as well as activation based on motivation and external stimuli.

We intend to present other papers about the methods of calculation of behavior value and the decomposition of behavior control parts into reflexive, common, and deliberative layers.

Acknowledgment

We would like to thank the entertainment robot company of Sony Coorporation and all of researchers and engineers of QRIO SDR-4XII.

References

[1] S. Ohnaka, T. Ando, and T. Iwasawa, "The introduction of the personal robot papero," *IPSJ SIG Notes*, vol. 2001, no. 68, pp. 37–42, 2001.

[2] H. Ishiguro, T. Ono, M. Imai, T. Kanda, and R. Nakatsu, "Robovie: an interactive humanoid robot," *Industrial Robot*, vol. 29, no. 6, pp. 498–503, 2001.

[3] C. Breazeal, Ed., *Designing Social Robots*. Cambridge: MIT Press., 2002.

[4] M. Fujita, Y. Kuroki, T. Ishida, and T.Doi, "Autonomous behavior control architecture of entertainment humanoid robot sdr-4x," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '03)*, Las Vegas, USA, Oct. 2003, pp. –.

[5] M. Fujita, K. Sabe, Y. Kuroki, T. Ishida, and T.Doi, "Sdr-4x ii: A small humanoid as an entertainer in home environment," in *11th International Symposiu of Robotics Research*, Siena, Italy, Oct. 2003, pp. –.

[6] M. Fujita and et al., "An autonomous robot that eats information via interaction with human and environment," in *The 10th IEEE International Workshop on Robot-Human Interactive Communication*, Bordeaux and Paris, France, Sept. 2001, pp. 383–389.

[7] R. Arkin, M. Fujita, T. Takagi, and R. Hasegawa, "Ethological and emotional basis for human-robot interaction," *Robotics and Autonomous System*, vol. 42, pp. 191–201, 2003.

[8] R.Arkin, M.Fujita, T.Takagi, and R.Hasegawa, "Ethological modeling and architecture for an entertainment robot," in *Proc. IEEE International Conference on Robotics and Automation (ICRA2001)*, Soel,Korea, May 2001, pp. –.