A Framework for Building Conversational Agents Based on a Multi-Expert Model

Mikio Nakano, Kotaro Funakoshi, Yuji Hasegawa, Hiroshi Tsujino
Honda Research Institute Japan Co., Ltd.
8-1 Honcho, Wako, Saitama 351-0188, Japan
{nakano, funakoshi, yuji.hasegawa, tsujino}@jp.honda-ri.com

Abstract

This paper presents a novel framework for building symbol-level control modules of animated agents and robots having a spoken dialogue interface. It features distributed modules called experts each of which is specialized to perform certain kinds of tasks. A common interface that all experts must support is specified, and any kind of expert can be incorporated if it has the interface. Several modules running in parallel coordinate the experts by accessing them through the interface, so that the whole system can achieve flexible control, such as interruption handling and parallel task execution.

1 Introduction

As much attention is recently paid to autonomous agents such as robots and animated agents, spoken dialogue is expected to be a natural interface between users and such agents. Our objective is to establish a framework for developing the intelligence module of such agents.

In establishing such a framework, we focus on achieving the following features. (1) *Multi-domain dialogue:* Since agents are usually expected to perform multiple kinds of tasks, they need to work in multiple domains and switch domains according to user utterances. (2) *Interruption handling:* It is crucial for human-agent interaction to be able to handle users' interrupting utterances while speaking or performing tasks. (3) *Parallel task execution:* Agents, especially robots that perform physical actions, are expected to be able to execute multiple tasks in parallel when possible. For example, robots should be able to engage in a dialogue while moving. (4) *Extensibility:* Since the agents can be used for a variety of tasks, various strategies for dialogue and task planning should be able to be incorporated.

Although a number of models for conversational agents have been proposed, no model has all of the above properties. Several multi-domain dialogue system models have been proposed and they are extensible, but it is not clear how they handle interruptions to system utterances and actions (e.g., O'Neill et al. (2004), Lin et al. (1999), and Hartikainen et al. (2004)). There are several spoken dialogue agents and robots that can handle interruptions thanks to their asynchronous control (Asoh et al., 1999; Boye et al., 2000; Blaylock et al., 2002; Lemon et al., 2002), they do not focus on making it easy to add new dialogue domains with a variety of dialogue strategies.

This paper presents a framework called RIME (Robot Intelligence based on Multiple Experts), which employs modules called *experts*.[1] Each expert is specialized for achieving certain kinds of tasks by performing physical actions and engaging in dialogues. It corresponds to the symbol-level control module of a system that can engage in tasks in a single small domain, and it employs fixed control strategies. Only some of the experts take charge in understanding user utterances and decide actions. The basic idea behind RIME is to specify a common interface of experts for coordinating them and to achieve flexible control. In RIME, several mod-

[1]RIME is an improved version of our previous model (Nakano et al., 2005), whose interruption handling was too simple and which could not achieve parallel task execution.

Proceedings of the 9th SIGdial Workshop on Discourse and Dialogue, pages 88–91,
Columbus, June 2008.

ules run in parallel for coordinating experts. They are *understander*, which is responsible for speech understanding, *action selector*, which is responsible for selecting actions, and *task planner*, which is responsible for deciding which expert should work to achieve tasks.

RIME achieves the above mentioned features. Multi-domain dialogues are possible by selecting an appropriate expert which is specialized to dialogues in a certain domain. Interruption handling is possible because each expert must have methods to detect interruptions and decide actions to handle interruptions, and coordinating modules can use these methods. Parallel task execution is possible because experts have methods for providing information to decide which experts can take charge at the same time, and the task planner utilizes that information. Extensibility is achieved because any kind of expert can be incorporated if it supports the common interface. This makes it possible for agent developers to build a variety of conversational agents.

2 Multi-Expert Model

This section explains RIME in detail. Fig. 1 depicts its module architecture.

2.1 Experts

Each expert is a kind of object in the object-oriented programming framework. In this paper, we call tasks performed by one expert *primitive tasks*. Experts should be prepared for each primitive task type. For example, if there is an expert for a primitive task type "telling someone's extension number", "telling person A's extension number" is a primitive task. By performing a series of primitive tasks, a complicated task can be performed. For example, a museum guide robot can perform "explaining object B" by executing "moving to B" and "giving an explanation on B". Among the experts, a small number of experts can perform tasks at one time. Such experts are called *being in charge*.

Each expert holds information on the progress of the primitive task. It includes task-type-independent information, such as which action in this primitive task is being performed and whether the previous robot action finished, and task-type-dependent information such as the user intention understanding results and dialogue history. The contents and the data structure for the task-type-dependent information for each expert can be designed by the system developer.

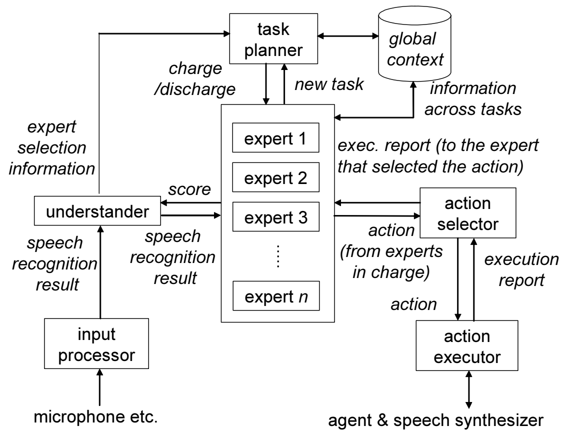

Figure 1: Architecture for RIME-Based Systems

Experts are classified into *system-initiative task experts* and *user-initiative task experts*. In this paper, the *initiative* of a task means who can initiate the task. For example, the task "understanding a request for weather information" is a user-initiative task, and the task "providing weather information" is a system-initiative task.

In RIME, executing multiple tasks in parallel becomes possible by making multiple experts take charge. To check whether two experts can take charge simultaneously, we currently use two features *verbal* and *physical*. Two experts having the same feature cannot take charge simultaneously.

The interface of experts consists of methods for accessing its internal state. Below are some of the task-type-dependent methods, which need to be implemented by system developers.

The *understand* method updates the internal state based on the user speech recognition results, using domain-dependent sentence patterns for utterance understanding. This method returns a score which indicates the plausibility the user utterance should be dealt with by the expert. Domain selection techniques in multi-domain spoken dialogue systems (Komatani et al., 2006) can be applied to obtain the score. The *select-action* method outputs one action based on the content of the internal state. Here, an *action* is a multimodal command which includes a text to speak and/or a physical action command.

The action can be an empty action, which means doing nothing. The *detect-interruption* method returns a Boolean value that indicates whether the previous user utterance is an interruption to the action being performed when this expert is being in charge. The *handle-interruption* method returns the action to be performed after an interruption is detected. For example, an instruction to stop the utterance can be returned.

In the definition of these methods, experts can access a common database called *global context* to store and utilize information across domains, such as information on humans, information on the environment, and past dialogue topics.

2.2 Modules Coordinating Experts

To exploit experts, three processes, namely the *understander*, the *action selector*, and the *task planner*, work in parallel.

The understander receives output of an *input processor*, which typically performs speech recognition. Each time the understander receives a user speech recognition result from the input processor, it performs the following process. First it dispatches the speech recognition result to the experts in charge and the user-initiative experts with their *understand* methods, which then returns the scores mentioned above. The expert that returns the highest score is selected as the expert to take charge. If the selected expert is not in charge, it tells the task planner that the expert is selected as the user-initiative expert to take charge. If the selected expert is in charge, it calls the *detect-interruption* method of the expert. If *true* is returned, it tells the action selector that an interruption utterance is detected.

The action selector repeats the following process for each expert being in charge in a short cycle. When an interruption for the expert is detected, it calls the expert's *handle-interruption* method, and it then sends the returned action to the action executor, which is assumed to execute multimodal actions by controlling agents, speech synthesizers, and other modules. Otherwise, unless it is not waiting for a user utterance, it calls the expert's *select-action* methods, and then sends the returned action to the action executor. The returned action can be an empty action. Note that it is assumed that the action executor can perform two or more actions in parallel when possible.

ID	task type	initiative	feature
A	understanding weather information requests	user	verbal
B	providing weather information	agent	verbal
C	understanding extension number requests	user	verbal
D	providing extension numbers	agent	verbal
E	understanding requests for guiding to places	user	verbal
F	moving to show the way	agent	physical
G	explaining places	agent	verbal

Table 1: Experts in the Example Robotic System

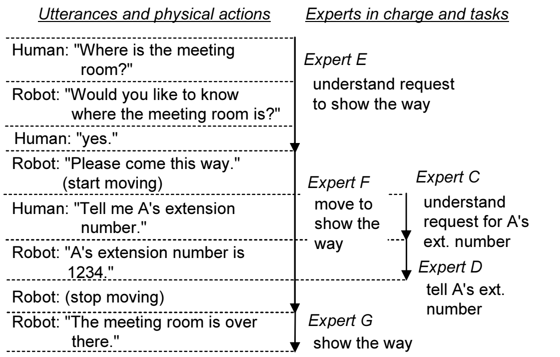

Figure 2: Expert Selection in a Parallel Task Execution Example

The task planner is responsible for deciding which experts take charge and which experts do not. It sometimes makes an expert take charge by setting a primitive task, and sometimes it discharges an expert to cancel the execution of its primitive task. To make such decisions, it receives several pieces of information from other modules. First it receives from the understander information on which expert is selected to understand a new utterance. It also receives information on the finish of the primitive task from an expert being in charge. In addition, it receives new tasks from the experts that understand human requests. The task planner also consults the global context to access the information shared by the experts and the task planner. In this paper we do not discuss the details of task planning algorithms, but we have implemented a task planner with a simple hierarchical planning mechanism.

There can be other processes whose output is written in the global context. For example, a robot and human localization process using image processing and other sensor information processing can be used.

3 Implementation as a Toolkit

The flexibility of designing experts increases the amount of effort for programming in building experts. We therefore developed RIME-TK (RIME-ToolKit), which provides libraries that facilitate building systems based on RIME. It is implemented in Java, and contains an abstract expert class hierarchy. The system developers can create new experts by extending those abstract classes. Those abstract classes have frequently used functions such as WFST-based language understanding, template-based language generation, and frame-based dialogue management. RIME-TK also contains the implementations of the understander and the action selector. In addition, it specifies the interfaces for the input processor, the action executor, and the task planner. Example implementations of these modules are also included in RIME-TK. Using RIME-TK, conversational agents can be built by creating experts, an input processor, an action executor, and a task planner.

As an example, we have built a robotic system, which is supposed to work at a reception, and can perform several small tasks such as providing extension numbers of office members and guiding to several places near the reception such as a meeting room and a restroom. Some experts in the system are listed in Table 1. Fig. 2 shows an example interaction between a human and the robotic system that includes parallel task execution and how experts are charged. The detailed explanation is omitted for the lack of the space.

By developing several other robotic systems and spoken dialogue systems (e.g., Komatani et al. (2006), Nakano et al. (2006), and Nishimura et al. (2007)), we have confirmed that RIME and RIME-TK are viable.

4 Concluding Remarks

This paper presented RIME, a framework for building conversational agents. It is different from previous frameworks in that it makes it possible to build agents that can handle interruptions and execute multiple tasks in parallel by employing experts which have a common interface. Although the current implementation is useful for building various kinds of systems, we believe that preparing more kinds of expert templates and improving expert selection for understanding utterances facilitate building a wider variety of systems.

Acknowledgments We would like to thank all people who helped us to build RIME-TK and its applications.

References

H. Asoh, T. Matsui, J. Fry, F. Asano, and S. Hayamizu. 1999. A spoken dialog system for a mobile office robot. In *Proc. Eurospeech-99*, pages 1139–1142.

N. Blaylock, J. Allen, and G. Ferguson. 2002. Synchronization in an asynchronous agent-based architecture for dialogue systems. In *Proc. Third SIGdial Workshop*, pages 1–10.

J. Boye, B. A. Hockey, and M. Rayner. 2000. Asynchronous dialogue management: Two case-studies. In *Proc. Götalog-2000*.

M. Hartikainen, M. Turunen, J. Hakulinen, E.-P. Salonen, and J. A. Funk. 2004. Flexible dialogue management using distributed and dynamic dialogue control. In *Proc. Interspeech-2004*, pages 197–200.

K. Komatani, N. Kanda, M. Nakano, K. Nakadai, H. Tsujino, T. Ogata, and H. G. Okuno. 2006. Multi-domain spoken dialogue system with extensibility and robustness against speech recognition errors. In *Proc. 7th SIGdial Workshop*, pages 9–17.

O. Lemon, A. Gruenstein, A. Battle, and S. Peters. 2002. Multi-tasking and collaborative activities in dialogue systems. In *Proc. Third SIGdial Workshop*, pages 113–124.

B. Lin, H. Wang, and L. Lee. 1999. Consistent dialogue across concurrent topics based on an expert system model. In *Proc. Eurospeech-99*, pages 1427–1430.

M. Nakano, Y. Hasegawa, K. Nakadai, T. Nakamura, J. Takeuchi, T. Torii, H. Tsujino, N. Kanda, and H. G. Okuno. 2005. A two-layer model for behavior and dialogue planning in conversational service robots. In *Proc. 2005 IEEE/RSJ IROS*, pages 1542–1547.

M. Nakano, A. Hoshino, J. Takeuchi, Y. Hasegawa, T. Torii, K. Nakadai, K. Kato, and H. Tsujino. 2006. A robot that can engage in both task-oriented and non-task-oriented dialogues. In *Proc. 2006 IEEE/RAS Humanoids*, pages 404–411.

Y. Nishimura, S. Minotsu, H. Dohi, M. Ishizuka, M. Nakano, K. Funakoshi, J. Takeuchi, Y. Hasegawa, and H. Tsujino. 2007. A markup language for describing interactive humanoid robot presentations. In *Proc. IUI-07*.

I. O'Neill, P. Hanna, X. Liu, and M. McTear. 2004. Cross domain dialogue modelling: an object-based approach. In *Proc. Interspeech-2004*, pages 205–208.