# Dynamic Conversation Structures: An Extended Example

Scott A. Moore

University of Michigan Business School, Ann Arbor, MI, USA,
samoore@umich.edu

**Abstract.** The author provides an in-depth look at a moderately complex conversation as represented by a finite state machine, a representation used by an established agent communication system. He compares this to a statechart-based method used for Moore's conversation policy framework and observes that these methods differ in their level of detail, the usefulness of parts of the graphical representation, and in the grouping of events. The remainder of the paper demonstrates how a multi-agent conversation policy can be used to control the flow of messages, contrasts this with how messages are handled via an inference-based process, and shows how the inference-based processing can be integrated with the policy-based handling in order to deal with exceptions to the policy.

## 1 Introduction

Researchers have proposed many agent communication languages (e.g., FIPA's ACL [Fou97], FLBC [Moo01], KQML [LF97]). In the development of each of these languages, it has become apparent that there needs to be a method for modeling conversation policies (see the collection in [DG00]). Moore defined a method of representing conversation policies, applied it to FLBC, and demonstrated it by modeling a series of relatively simple conversations [Moo00b]. In this paper I apply this method to a more complex conversation. This example shows how this method works and how the system can be used to coordinate the actions of two conversational participants.

A conversation among agents is composed of an exchange of messages. This exchange is generally aims at the accomplishment of some task or the achievement of some goal. In its simplest form it is a sequence of messages in which, after the initial message, each is a direct response to the previous one. More complicated structures occur when subdialogs are needed. Linguists and philosophers have not come to any consensus about subdialogs, but several types consistently appear:[1] subordinations, corrections, and interruptions. A message begins a *subordinate* conversation when it elaborates on a point made in a previous message. This message should be about the previous message, probably as a clarification of some fact. A message that begins a *correction*

[1] See [LA87,Moo01,Pol88].

subdialog indicates that the message somehow corrects a previous message in the conversation. A message *interrupts* the current conversation when it is neither an expected nor the standard reply to the previous message. This should be used only if the other two do not apply.

A conversation policy (CP) defines 1) how one or more conversation partners respond to messages they receive, 2) what messages one partner expects in response to a message it sends, and 3) the rules for choosing among competing courses of action. These policies can be used both to describe how a partner will respond to a message (or series of messages) it receives and to specify the procedure the partner actually executes in response to that message (or those messages). The policy is a template describing an agent's reactions to an incoming message, its expectations about upcoming messages, and the rules it applies to determine its own reactions. In a linguistic sense, it moves the conversation from an inference-based process to a convention-based one. Inference acts as a backup, providing a more computationally expensive way of understanding unfamiliar messages for which CPs have not been defined. The means by which this backup can be implemented is demonstrated in §2.

Conversation policies are represented using the statechart formalism defined by Harel [Har87] and then later integrated into the UML.[2] This is a graphical language which developers should find easier to work with than, e.g., an underlying XML representation which the agents themselves might use. Statecharts are quite appropriate for event-driven systems, conveniently representing "the set of allowed sequences of input and output events, conditions, and actions, perhaps with some additional information such as timing constraints." [Har87, pp. 231–2].

Since CPs are used to govern an agent's behavior in a conversation, each one begins with a message being either sent or received. The sent or received message acts as a tag for the CP, indicating when it will be invoked. An agent maintains two databases related to CPs. One is the set of CPs it can use and the other is the set of currently executing CPs. Figure 1 contains a sample statechart representation of a conversation policy. The whole state-

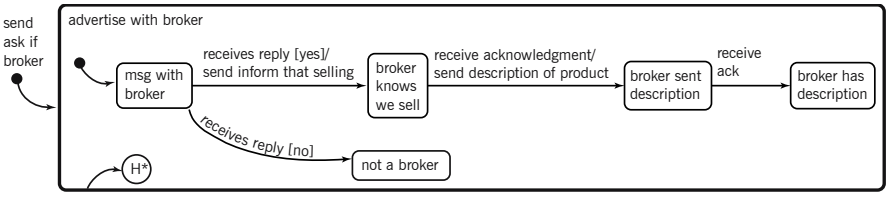


**Fig. 1.** A sample statechart representation of a conversation policy

[2] [BRJ99,RJB99]