Proceedings of the Third SIGdial Workshop on Discourse and Dialogue,
Philadelphia, July 2002, pp. 60-63. Association for Computational Linguistics.

A Flexible Framework for Developing Mixed-Initiative Dialog Systems

Judith HOCHBERG, Nanda KAMBHATLA, Salim ROUKOS
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598, USA
{judyhoch, nanda, roukos}@us.ibm.com

Abstract

We present a new framework for rapid development of mixed-initiative dialog systems. Using this framework, a developer can author sophisticated dialog systems for multiple channels of interaction by specifying an interaction modality, a rich task hierarchy and task parameters, and domain-specific modules. The framework includes a dialog history that tracks input, output, and results. We present the framework and preliminary results in two application domains.

1 Introduction

Developing a mixed-initiative dialog system is a complex task. The developer must model the user's goals, the "results" (domain objects) retrieved, and the state of the dialog, and generate the system response at each turn of the dialog. In mixed-initiative systems, as opposed to directed dialog systems, users can influence the dialog flow, and are not restricted to answering system questions in a prescribed format (e.g. Walker 1990, Chu-Carroll 2000).

Compounding these challenges, dialog applications have evolved from simple look-up tasks to complex transactional systems like telephony banking and stock trading (Zadrozny et al. 1998), and air travel information systems. These systems increasingly cater to multiple channels of user interaction (telephone, PDA, web, etc.), each with its own set of modalities. To simplify the development of such systems, researchers have created frameworks that embody core dialog functionalities.

In MIT's framework, a developer creates a dialog system by specifying a dialog control table comprising actions and their triggering events. The developer has great freedom in designing this table, but must specify basic actions such as prompting for missing information. As a result, these tables can become quite complex – the travel system control table contains over 200 ordered rules. MIT has applied this framework to both weather and travel (Zue et al. 2000, Seneff and Polifroni 2000).

In IBM's form-based dialog manager, or FDM (Papineni et al. 1998), a developer defines a set of forms that correspond to separate tasks in the application, such as finding a flight leg. The forms have powerful built-in capabilities, including mechanisms that trigger various types of prompts, and allow the user to specify inheritance and other relationships between tasks. Just as in the MIT framework, domain-specific modules perform database queries and other backend processes; the forms call additional developer-defined modules that affect the dialog state and flow. FDM has supported dialog systems for air travel (Papineni et al. 1999, Axelrod 2000) and financial services (IBM 2001, IBM 2002). The University of Colorado framework also has a form-based architecture (Pellom et al. 2001), while CMU and Bell Labs' frameworks allow the specification of deep task hierarchies (Wei and Rudnicky 2000, Potamianos et al. 2000).

Our goal is to design a framework that is both powerful, embodying much dialog functionality, and flexible, accommodating a variety of dialog domains, modalities, and styles. Our new framework goes beyond FDM in building more core functionality into its task model, yet provides a variety of software tools, such as API calls and overwritable functions, for customizing tasks. The framework allows developers to specify a wide range of relationships among tasks, and provides a focus model that respects these relationships. To support the task framework we introduce a

dialog history component that remembers input, output, and cumulative task results. Section 2 of this paper describes the framework, and section 3 some applications. In section 4 we discuss future plans and implications.

2 The HOT framework

Our framework's moniker is HOT, which stands for its three components: dialog **H**istory, domain **O**bjects, and **T**ask hierarchy. It is implemented as a Java library. In this section, we describe the HOT framework. We assume the existence of an application specific natural language parser that brackets and labels chunks of text corresponding to domain specific attributes, and a natural language generation module for generating prompts from abstract specifications.

2.1 Task hierarchy

A task defines a unit of work for a dialog system. The HOT framework enables the specification of tasks that are organized as a hierarchy (e.g. Fig. 1). The terminal tasks in the hierarchy (UserID, Fund, and Shares) derive canonical values of domain attributes (such as fund symbol) from parsed portions of user input. The RootTask specifies methods for managing the dialog, e.g. for disambiguating among different sub-tasks in case of ambiguous user input. All other tasks perform scripted actions using the output produced by other non-terminal or terminal tasks: generate a user profile, a share transaction, or a price quote.

The task hierarchy constitutes a plan for the dialog. It remains to be seen whether it can also be used for planning domains in which task input can come either from a user or from an external process such as an environmental monitor, as in [Allen at al. 2001].

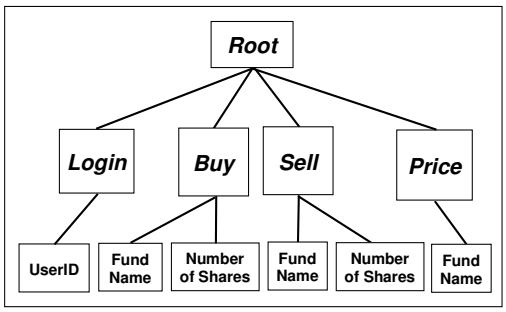

Figure 1: A task hierarchy for a simple mutual fund application.

The framework allows developers to easily specify five different relationships among tasks in a hierarchy. Many of these will be exemplified in Section 3.

1. *Subtasking*: UserID is a subtask of Login because Login needs the user's ID to log the user in.
2. *Ordering*: Login precedes all other tasks, but Buy, Sell, and Price are unordered.
3. *Cardinality*: Login is executed only once per session, and UserId, Fund, and Shares are executed only once per parent task. However, Buy, Sell, and Price can be executed multiple times.
4. *Inheritance*: Buy and Sell can potentially inherit a fund name from Price and vice versa.
5. *Subdialog*: The user can carry out certain subdialogs, such as a Price query within a Buy task.

2.2 Focus model

At each turn of the dialog, we automatically score the user's input to infer the task that the user wants to work on. Only a non-terminal task can receive focus. As in FDM, scoring is primarily based on the number of matches between attributes in the parsed user input, different task attributes, and the last system prompt. The developer can specify the appropriate system behavior if the inferred user focus conflicts with task relationships, e.g. if a user wants to Buy but has not yet Logged in. In the absence of such conflicts, the framework triggers execution of the inferred task. If the task completes without ending a turn, the focus model returns focus to a previously started task if possible, or else defaults to the developer's preference for what to do next.

2.3 Task functionality

Within RootTask, a developer can specify the modalities of interaction and the specific backends used, create an initial task layout, and set some dialog parameters. Developers must specify how they want RootTask to respond to various focus situations. For example, if no tasks are eligible for focus, this may represent an error condition in one application, but the expected end of a dialog in another application.

For all other tasks, task functionality can be divided into operations that happen before and after the task calls its backend process

(accessing a database, the Internet, or other information channel) to create a result. Pre-backend functionality involves assessing, and possibly confirming with the user, the parameters to be sent to the backend. Post-backend functionality acts on different backend outcomes: for example, informing the user of a result, confirming a result, or requesting further constraints. Because the framework already defines these functionalites, the developer's role is to define the backend and its result, and to choose the pre-defined functionalities that apply.

As tasks execute, they post communicative intentions – dialog acts (e.g., "Inform", "Confirm") and the domain objects they concern (e.g., flights) – to the dialog history. A separate NLG module generates the text of the system response based on these communicative intentions and the specific modalities in use.

2.4 Dialog History

The dialog history provides developers with an organized way to store system data regardless of the application domain. We store the user input (attribute-value pairs), the system response (communicative intentions), and the cumulative results for each dialog turn. The developer can additionally store the user input at various stages of processing. Results can be generic objects (e.g., a String representing a fund name) or complex, domain-specific objects. The results are tagged with a core set of status indicators that developers can add to. The dialog history also serves as a task stack, since the current result from each task indicates the task's status.

The dialog history is reminiscent of BBN's state manager (Stallard 2000), but the latter also includes many dialog management responsibilities that we reserve to tasks.

3 Applications

We are currently implementing the HOT framework described above. In this section, we describe two sample applications built using this framework as implemented so far.

3.1 Mutual Funds

We built a mutual funds application, using the framework to create a task hierarchy and to add task relationships such as ordering and inheritance, as described in section 2.1. We customized the tasks by writing backend methods and specifying confirmation strategies, the domain attributes of interest to each task, and domain-specific result objects (UserProfile and ShareTransaction). Figure 2 shows an actual dialog with this application.

The dialog flows through a combination of developer specifications and the HOT framework, with the framework doing most of the work. In turn 1, only Login is specified as active, so the framework generates an initial prompt for it. In turn 2, the framework scores the input and executes both UserID and Signin. It informs the user of Login's success (as specified), then issues a general prompt because Login enabled all other tasks when it finished (as specified). In turn 3, the framework issues an error message because of the conflict between Login's cardinality of 1 and the user's input. In turn 4, the fund name lookup finds an ambiguous result; the framework looks to the developer to specify what type of prompt to issue in response. In turn 5, the framework correctly interprets the user's selection, and requests fund name confirmation (as specified). In turn 6, Sell runs its Number of Shares subtask. In turn 7, the user performs an authorized Price subdialog from within Sell. Price inherits the fund name from Sell, executes the task, and returns focus to Sell. In turn 8, the framework requests confirmation prior to

	User	System
1		Welcome to HOT-DM. User ID?
2	1677	Greetings Judy Hochberg! How may I help you?
3	I want to sign in again.	Sorry, you can only sign in once. How may I help you?
4	I want to sell Watson fund	Watson Yorktown fund or Watson Hawthorne fund?
5	the first one	Is Watson Yorktown the fund you want to sell?
6	yes	How many shares?
7	can you tell me what the price is	The price of Watson Yorktown is $100.2 per share How many shares?
8	I want to sell 30 shares	Do you want to sell 30 shares of Watson Yorktown fund?
9	yes i do	Transaction #1: sold 30 shares of Watson Yorktown fund at $100.2 per share on 2/18/2002 How may I help you?

Figure 2: A dialog with a mutual funds application built using the HOT framework.

executing the sale (as specified). In turn 9, note that the Sell backend accesses the dialog history to determine the correct transaction number.

3.2 Travel

We have developed a travel information dialog system using the HOT framework. For this application, we reused several components of the IBM DARPA Communicator system. The system accepts either standard text input or text from instant messaging from a mobile platform. Figure 3 shows a screen shot of a dialog on a "Palm" (PDA) interface to the travel system.

4 Discussion

We have presented a new framework for developing mixed-initiative dialog systems. This framework, dubbed HOT, enables developers to rapidly develop dialog systems by specifying tasks, their relationships, and relevant domain objects. We are currently implementing this framework as a toolkit and have developed two sample applications in two different modalities.

The new framework departs from other frameworks in the range of functionality that it covers. Its task model triggers not only informational prompts and confirmations, but also customizable responses to task problems of different sorts, such as underspecification. The task relationships modeled are likewise quite rich, including subdialog and inheritance. Finally, the dialog history provides a generic specification of output semantics, a way to track task status, and uniform access to dialog results of varying complexity. Our future goal is continue to build functionality, especially in NLG, without sacrificing flexibility.

Figure 3: A dialog in a "Palm" interface to an air travel dialog system.

References

J. Allen, G Ferguson, and Amanda Stent (2001) *An architecture for more realistic conversational systems*. Proc. Intelligent User Interfaces.

S. Axelrod, (2000) *Natural Language Generation in the IBM Flight Information System*. Proc. ANLP-NAACL Workshop on Conversational Systems.

J. Chu-Carroll (2000) *MIMIC: An Adaptive Mixed Initiative Spoken Dialogue System for Information Queries*. Proc ANLP.

IBM (2001) http://www-3.ibm.com/software/speech/news/20010609trp.html

IBM (2002) http://www-3.ibm.com/software/speech/enterprise/dcenter/demo_2.html

K. Papineni, S. Roukos, and T. Ward (1999) *Free-Flow Dialog Management Using Forms*. Proc. Eurospeech, pp. 1411-1414.

B. Pellom, W. Ward, J. Hansen, K. Hacioglu, J. Zhang, X. Yu, and S. Pradhan (2001) *University of Colorado Dialog Systems for Travel and Navigation*. Proc. HLT.

A. Potamianos, E. Ammicht, and H-K. Kuo (2000) *Dialogue Management in the Bell Labs Communicator System*. Proc. ICSLP.

S. Seneff and J. Polifroni (2000) *Dialogue Management in the Mercury Flight Reservation System*. Proc. Satellite Dialogue Workshop, ANLP-NAACL.

D. Stallard (2000) *Talk'N'Travel: A Conversational System for Air Travel Planning*. Proc ANLP.

M. Walker (1990) *Mixed Initiative in Dialogue: An Investigation into Discourse Segmentation.* Proc. ACL90, pp. 70-78.

X. Wei and A. Rudnicky (2000) *Task-based dialog management using an agenda*. Proc ANLP/NAACL Workshop on Conversational Systems, pp. 42-47.

V. Zue, S. Seneff, J. Glass, J. Polifroni, C. Pao, T. Hazen, and L. Hetherington (2000) *JUPITER: A Telephone-Based conversational Interface for Weather Information*. IEEE Trans. Speech and Audio Proc., 20/Y, pp. 100-112.

W. Zadrozny, C. Wolf, N. Kambhatla, and Y. Ye, 1998. *Conversation Machines for Transaction Processing*. PROC AAAI/IAAI, pp. 1160-1166.