Dialogue Management Based on Entities and Constraints

Yushi Xu **Stephanie Seneff**
Spoken Language Systems Group
MIT Computer Science and Artificial Intelligence Laboratory
United States
{yushixu, seneff}@csail.mit.edu

Abstract

This paper introduces a new dialogue management framework for goal-directed conversations. A declarative specification defines the domain-specific elements and guides the dialogue manager, which communicates with the knowledge sources to complete the specified goal. The user is viewed as another knowledge source. The dialogue manager finds the next action by a mixture of rule-based reasoning and a simple statistical model. Implementation in the flight-reservation domain demonstrates that the framework enables the developer to easily build a conversational dialogue system.

1 Introduction

Conversational systems can be classified into two distinct classes: goal-directed and casual chatting. For goal-directed systems, the system is usually more "knowledgeable" than the user, and it attempts to satisfy user-specified goals. The system's conversational strategies seek the most efficient path to reach closure and end the conversation (Smith, Hipp, & Biermann, 1995).

An essential commonality among different goal-directed applications is that, at the end of a successful conversation, the system presents the user with a "goal" entity, be it a flight itinerary, a route path, or a shopping order. Different conversations result from different properties of the goal entities and different constraints set by the knowledge sources. The properties define the necessary and/or relevant information, such as flight numbers in the flight itinerary. Constraints specify the means to obtain such information. For examples fields "source", "destination" and "date" are required to search for a flight. Once the properties and constraints are known, dialogue rules can easily map to dialogue actions.

This paper introduces a dialogue management framework for goal-directed conversation based on entity and knowledge source specification. The user is viewed as a collaborator with the dialogue manager, instead of a problem-raiser. The dialogue manager follows a set of definitions and constraints, and eventually realizes the goal entity. It also incorporates a simple statistical engine to handle certain decisions.

2 Related Work

In recent years, statistical methods have gained popularity in dialogue system research. Partially Observable Markov decision processes have been the focus of a number of papers (Levin, Pieraccini, & Eckert, 1997; Scheffler & Young, 2001; Frampton & Lemon, 2006; Williams & Young, 2007). These approaches turn the dialogue interaction strategy into an optimization problem. The dialogue manager selects actions prescribed by the policy that maximizes the reward function (Lemon & Pietquin, 2007). This machine learning formulation of the problem automates system development, thus freeing the developers from hand-coded rules.

Other researchers have continued research on rule-based frameworks, in part because they are easier to control and maintain. One common approach is to allow developers to specify the tasks, either using a conditioned sequential script (Zue, et al., 2000; Seneff, 2002), or using a task hierarchy (Hochberg, Kambhatla, & Roukos, 2002). In (Bohus & Rudnicky, 2003)'s work, a tree of dialogue agents, each of which handles different dialogue actions, is specified to control the dialogue progress. The knowledge has also been specified either by first order logic (Bühler & Minker, 2005) or ontology information (Milward & Beveridge, 2004).

3 Dialogue Manager

Figure 1 illustrates the architecture of the proposed dialogue management framework. Com-

Proceedings of SIGDIAL 2010: the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue, pages 87–90,
The University of Tokyo, September 24-25, 2010.

munication with the dialogue manager (DM) is via "E-forms" (Electronic forms), which consist of language-independent key-value pairs. The language understanding and language generation components mediate between the DM and various knowledge sources (KS), including the user, to interpret the output from the KS and generate input that the KS can understand. Each KS handles one or more sub-domains. For example, a date/time KS can resolve a date expression such as "next Tuesday" to a unique date; a flight database can provide flight information. The KSes are provided by the developer. They can be local (a library) or external (a separate executable).

Within this architecture, the user is viewed as a special KS, who understands and speaks a natural language, so that the whole architecture is completely DM-centered, as shown in Figure 1. An external language understanding system parses the original input into an E-form, and an external language generation component converts the output E-form into the desired natural language. Each particular communication with the user is analogous to other communications with the various KSes. The user is always ranked the lowest in the priority list of the KSes, *i.e.*, only when other knowledge sources cannot provide the desired information does the DM try to ask the user.

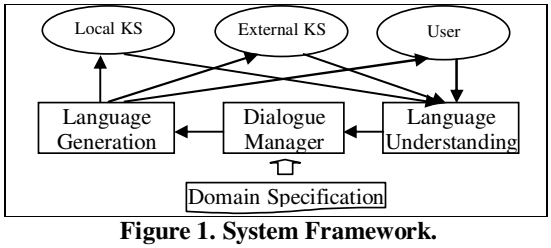

Figure 1. System Framework.

For example, in the flight reservation system, suppose the DM first tries to determine the source airport. If there exists a KS that contains this user's home airport information, the DM will adopt it. If no other KS can provide the information, the DM asks the user for the departure city.

3.1 Entity-Based Specification

Our framework uses an entity-based declarative domain specification. Instead of providing the action sequence in the domain, the developer provides the desired form of the goal entity, and the relationships among all relevant entities.

The specification is decomposed into two parts. The first part is the declaration of the knowledge sources. Each KS may contain one or more sub-domains, and an associated "nation" defines the language processing parameters.

The second part is the entity type definition. For a particular domain, there is one goal entity type, and an arbitrary number of other entity types, *e.g.*, two entity types are defined in the flight reservation system: "itinerary" and "flight." The definition of an entity type consists of a set of members, including their names, types and knowledge domain. A logical expression states the conditions under which the entity can be regarded as completed; *e.g.*, a completed itinerary must contain one or more flights. The entity definition can also include optional elements such as comparative/superlative modifiers or customized command-action and task-action mappings, described in more detail later.

The entity-based specification has an advantage over an action-based specification in two aspects. First, it is easier to define all the entities in a dialogue domain than to list all the possible actions, so the specification is more compact and readable. Secondly, the completion condition and the KS's constraints capture the underlying motivation of the dialogue actions.

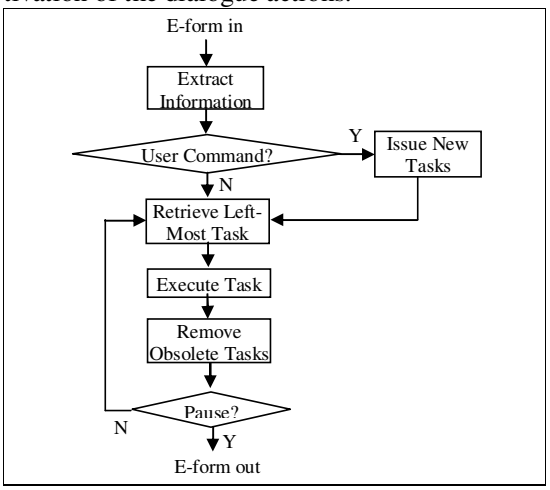

Figure 2. The Main Loop of the DM.

3.2 Dialogue Execution

Similar to the Information-State-Update (Larsson & Traum, 2000) idea, the DM maintains an internal state space with all up-to-date information about the entities. It also keeps a task list tree with a root task "complete goal." In task execution, subtasks (child node) and/or subsequent (right sibling node) tasks are issued. Each time the left-most leaf task is executed, and when a task is completed, the DM checks all tasks and removes those that have been rendered obsolete.

Ten basic tasks are pre-defined in the DM, including *complete_entity*, *inquire_ks*, and some other tasks related to entity manipulation. A *complete_entity* task evaluates the completion

conditions and issues appropriate tasks if they are unmet. An *inquire_ks* task handles communication with the KSes, and issues subtasks if the query does not satisfy the constraints. A default action associated with each task can be replaced by customized task-action mappings if needed.

Figure 2 shows the main loop of the DM. The process loops until a "pause" is signaled, which indicates to await the user's spoken response. An example will be given in Section 4.

3.3 Statistical Inference

To cope with situations that rules cannot handle easily, the framework incorporates a simple statistical engine using a Space Vector Model. It is designed only to support inference on specific small problems, for example, to decide when to ask the user for confirmation of a task. Models are built for each of the inference problems. The output label of a new data point is computed by weighting the labels of all existing data by their inverse distances to the new data point.

Equations (1) to (3) show the detailed math of the computation, where x is the new data point and d^j is the j-th existing data point. α is a fading coefficient which ranges from 0 and 1. β, a correction weight, has a higher value for data points resulting from manual correction. $\delta(\cdot)$ is 1 when the two inputs are equal and 0 otherwise. *sim(x, d)* defines the similarity between the new data point and the existing data point. Function $dis(\cdot)$ indicates the distance for a particular dimension, which is specified by the developer. The weight for each dimension w_i is proportional to the count of distinct values of the particular dimension $c(\boldsymbol{D}_i)$ and the mutual information between the dimension and the output label.

$$f(x) = \underset{y_i}{\operatorname{argmax}} \sum_j \alpha_j \beta_j sim(x, d^j) \cdot \delta(f(d^j), y_i) \quad (1)$$

$$\boldsymbol{sim}(\boldsymbol{x}, \boldsymbol{d}) = \begin{cases} \dfrac{\mathbf{1}}{\sqrt{\sum_i \boldsymbol{w_i} \cdot \boldsymbol{dis^2}(\boldsymbol{x_i}, \boldsymbol{d_i})}} & \boldsymbol{x \neq d} \\ \boldsymbol{S} & \boldsymbol{x = d} \end{cases} \quad (2)$$

$$w_i \propto c(\boldsymbol{D}_i) H(\boldsymbol{D}_i, f(\boldsymbol{D})) \quad (3)$$

4 Implementation in Flight Domain

The framework has been implemented in the flight reservation domain. A grammar was used to parse the user's input, and a set of generation rules was used to convert the DM's output E-form into natural language (Seneff, 2002). Two local KSes are utilized: one resolves complex date and time expressions, and one looks up airport/city codes. A local simulated flight DB will be replaced by a real external one in the future.

Figure 3 illustrates the logic of the flight reservation domain. The database has two alternative sets of conjunctive constraints "destination & source & date" and "flight# & date". Two entity types are defined. The itinerary entity type contains a list of flights, a number of expected flights and a price, with completion condition "#flights > 0". The flight entity type contains members: flight number, date, source, destination, *etc.*, with completion condition "flight# & date".

Table 1 illustrates dialogue planning. In the execution of *flight.complete_entity()*, the DM determines that it needs a flight number according to the entity's completion condition. However, a destination is required to search the flight DB. No other KS offers this information, so the system turns to the user to ask for the destination.

The statistical engine currently supports inference for two problems: whether the execution of a task requires the user's confirmation, and whether the pending list is in focus.

Several customized task actions were defined for the domain. For example, after adding the first flight, a customized task action will automatically create a return flight with appropriate source and destination, unless a one-way trip has been indicated. The implementation of the customized task actions required only about 550 lines of code.

User: I want a flight to Chicago
create itinerary
itinerary.complete_entity()
itinerary.add_entity(:flights)
create flight
flight.complete_entity()
flight.fill_attribute(flight#)
inquire_ks(flight_db, flight#)
flight.fill_attribute(destination)
inquire_ks(user, destination)
System: What city does the flight leave from?

Table 1. An example of the system's reasoning process. Shaded lines denote statistical decisions.

5. Preliminary Evaluation

We conducted a preliminary evaluation with a simulated flight database and a simulated user model. The statistical inference model was trained with 210 turns from 18 conversations. A personality-based user simulator creates random scenarios and simulates user utterances using a probabilistic template-based method. In 50 conversations between the simulated user and the DM, the average number of turns was 14.58, with a high standard deviation of 8.2, due to the variety of the scenario complexity and personalities of the simulator users. Some simulated users

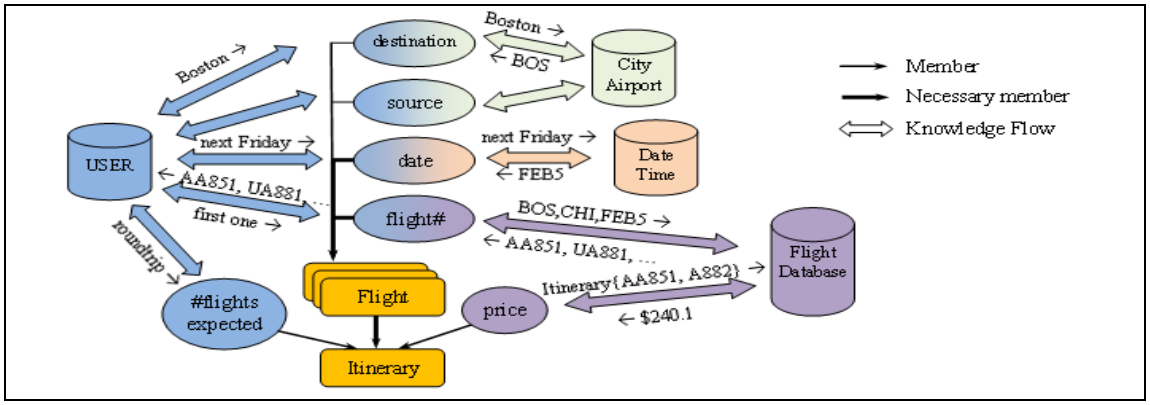

Figure 3. Dialogue Logic for the Flight Booking Domain.

were intentionally designed to be very uncooperative. The DM was able to handle these situations most of the time.

We examined all the simulated dialogues turn by turn. For a total of 729 turns, the DM responded appropriately 92.2% of the time. One third of the failed turns were due to parse failures. Another third resulted from insufficient tutoring. These situations were not well covered in the tutoring phase, but can be easily fixed through a few more manual corrections. The rest of the errors came from various causes. Some were due to defects in the simulator.

6 Conclusions and Future Work

We have introduced a framework for goal-based dialogue planning. It treats the user as a knowledge source, so that the entire framework is DM-centered. A declarative entity-based specification encodes the domain logic simply and clearly. Customized task actions handle any domain-dependent computations, which are kept at a minimum. A simple statistical engine built into the framework offers more flexibility.

In the future, we will integrate the dialogue manager into a speech-enabled framework, and build spoken dialogue systems for flight reservations and other domains of interest.

Acknowledgments

This research is funded by Quanta Computers, Inc., through the T-Party project.

References

Bohus, D., & Rudnicky, A. I. (2003). RavenClaw: Dialog Management Using Hierarchical Task Decomposition and an Expectation Agenda. *Proc. Eurospeech.* Geneva, Switzerland.

Bühler, D., & Minker, W. (2005). A REASONING COMPONENT FOR INFORMATION-SEEKING AND PLANNING DIALOGUES. *Spoken Multimodal Human-Computer Dialogue in Mobile Environments , 28*, 77-91.

Frampton, M., & Lemon, O. (2006). Learning more effective dialogue strategies using limited dialogue move features. *Proc. ACL*, (pp. 185 - 192). Sidney, Australia.

Hochberg, J., Kambhatla, N., & Roukos, S. (2002). A flexible framework for developing mixed-initiative dialog systems. *Proc. the 3rd SIGdial workshop on Discourse and dialogue* , (pp. 60-63). Philadelphia, Pennsylvania .

Larsson, S., & Traum, D. (2000). Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering , 6* (3-4), 323-340.

Lemon, O., & Pietquin, O. (2007). Machine learning for spoken dialog systems. *Proc. INTERSPEECH 2007*, (pp. 2685–2688). Antwerp, Belgium.

Levin, E., Pieraccini, R., & Eckert, W. (1997). Learning Dialogue Strategies within the Markov Decision Process Framework. *Proc. ASRU 1997.* Santa Barbara, USA.

Milward, D., & Beveridge, M. (2004). Ontologies and the Structure of Dialogue. *Proc. of the Eighth Workshop on the Semantics and Pragmatics of Dialogue*, (pp. 69-76). Barcelona, Spain.

Scheffler, K., & Young, S. (2001). Corpus-based dialogue simulation for automatic strategy learning and evaluation. *Proc. NAACL Workshop on Adaptation in Dialogue.* Pittsburgh, USA.

Seneff, S. (2002). Response Planning and Generation in the Mercury Flight Reservation System. *Computer Speech and Language , 16*, 283-312.

Smith, R. W., Hipp, D. R., & Biermann, A. W. (1995). An architecture for voice dialog systems based on prolog-style theorem proving. *Computational Linguistics , 21* (3), 281-320.

Williams, J. D., & Young, S. (2007). Partially observable Markov decision processes for spoken dialog systems. *Computer Speech & Language , 21* (2), 393-422.

Zue, V., Seneff, S., Glass, J., Polifroni, J., Pao, C., Hazen, T. J., et al. (2000). JUPITER: a telephone-based conversational interface for weather information. *IEEE Transactions on Speech and Audio Processing , 8* (1), 85-96.