SPOKEN LANGUAGE DIALOGUE: FROM THEORY TO PRACTICE

Esther Levin, Roberto Pieraccini,
Wieland Eckert, Pino Di Fabbrizio, Shrikanth Narayanan
{esther,roberto,eckert,pino,shri}@research.att.com

AT&T Labs-Research, 180 Park Avenue
Florham Park, NJ

ABSTRACT

A spoken language dialogue system is composed of many parts: speech recognition, speech synthesis, natural language understanding, dialogue manager, database, etc. Building a spoken language system for a new application requires in general a big effort for integrating all these parts, in addition to the effort required for designing, testing and tuning the dialogue behavior. This is especially true when complex dialogue tasks are built based on a mixed initiative paradigm. The approach suggested in this paper consists in shaping both the overall general architecture as well as the particular dialogue strategy as sequential decision processes. We discuss the concepts of stateful and stateless dialogue managers and we introduce a scripting language, DMD, which supports both. Finally we conclude with the description of the dialogue manager implemented for the DARPA Communicator multi leg travel task.

1. DIALOGUE AS A SEQUENTIAL DECISION PROCESS

A dialogue system can be formalized as a sequential decision process in terms of its *state space*, the set of possible *actions*, and the *strategy* [5][7]. The *state space* is defined by the collection of all variables that characterize the state of the dialogue system at a certain point in time. The *set of actions* describes what the system can do, i.e. the set of functions the system can invoke at any time (e.g. play a certain prompt, query a database, hang up, etc.). The *strategy* is a mapping between the state space and the action set. For any possible state the strategy prescribes what is the next action to perform. As a result of the action and its interaction with the external environment (e.g. user, database, etc.) the system gets some new observations (e.g. output of ASR, database tuples, etc.). The new observations are registered and modify the state of the system. This process continues until a final state is reached (e.g. the state after hang up in a telephone interaction). Following this formalization, the process of a dialogue system can be summarized by the algorithm in Fig. 1. The system starts in the initial state S_I. S_t denotes the system state at turn *t*. The function *NextAction* determines the next action A_t to be invoked, and the function *NextState* updates the state variables with the external observations. The process is repeated until a final state S_F is reached.

$S_t = S_I$
while $S_t \neq S_F$ {
  $A_t = \text{NextAction}(S_t)$
  invoke A_t
  O_t = environment response to A_t
  $S_{t+1} = \text{NextState}(S_t, A_t, O_t)$
  $t = t + 1$
}

Figure 1: Dialogue as a sequential process

2. THE AMICA DIALOGUE ARCHITECTURE

In this section we show how the sequential decision process formalization was implemented in the AT&T dialogue architecture and toolkit (AMICA [8]).

2.1 The Dialogue State

The dialogue state in the AMICA architecture is represented by a recursive key/value data structure. We call this structure *template*. A template can be easily mapped to similar data structures such as the Galaxy *frames* [6] used by MIT and constituting the standard for the DARPA Communicator [9] architecture.

2.2 Dialogue Actions

The action set of the dialogue system includes all possible actions it can perform, such as interactions with the user (e.g. asking the user for input, providing a user some output, confirmations, etc.), interactions with other external resources (e.g. querying a database), and internal processing. In our architecture an action is a function that reads a template, possibly interacts with the environment, and returns a modified template.
Of course actions can be defined at different levels of granularity. For example an action corresponding to the interaction with a user can be broken into separate lower level actions, including generating the prompt, activating the speech synthesizer, activating the speech recognizer with appropriate grammars, getting input from the speech recognizer, and parsing it.
It would be desirable to have a small set of generic high level parametric dialogue actions that could be used in any dialogue system. Indeed we can define a set of high-level abstract dialogue actions, or dialogue acts, such as confirmation, constraining,

relaxation, etc. But our experience showed that it is not possible to achieve a satisfactory application independent parameterization of such abstract high level dialogue actions. In fact any application requires fine-tuning of the strategy, and it is not possible to define a reasonably simple parametric form of dialogue actions that will allow for that. The solution we adopted consists of implementing a set of elemental actions (e.g. Input, Output, simple state editing operations, flow control, etc.) that can be used as building blocks for more complex actions. A scripting language (DMD) was developed both for building dialogue actions as well as for scripting the strategy.

2.3 DMD – the Dialogue Manager Developer scripting language

DMD is a scripting language that can be used both for building dialogue actions as well as for building the dialogue strategy. Other dialogue actions (see [8]) can be invoked within a DMD script. The interpreter reads the initial state of the dialogue, executes the script, and returns the final state before the process ends. Therefore a DMD script conforms to our definition of dialogue action, and can be invoked as a higher level action by another script. DMD also includes flow control statements (if, while, for, foreach, etc.) conditioned on the current state that are used to specify the dialogue strategy.

The advantages of DMD are:
- the concept of state is embedded in the language.
- DMD allows the use of high level operations.
- a DMD script can run either in *stateless* or *stateful* mode.

2.3.1 Stateful and Stateless modes

A dialogue manager can often be used in architectures that support a persistent connection between it and the rest of the modules. This corresponds to a *stateful* operation, i.e. the dialogue manager process will exist for the whole duration of the interaction, and it will maintain the evolving dialogue state at any time from the beginning to the end of the current session. This situation is summarized by Fig. 2a. The dialogue process is in state S_t. When input I_t is received by some other module, the dialogue process generates the output O_t while changing its internal state to S_{t+1}.

However, in several architectural situations it is not possible to guarantee the persistence of the dialogue process. For example in a web interaction based on the *cgi* protocol, for each single turn of interaction with a client browser a new dialogue process is started on the web server. Another example is the MIT Galaxy Hub architecture [6][9] on which the DARPA Communicator process is based. The Hub is a central router that allows communication among a set of (possibly) stateless servers. The Hub is the only process in the whole architecture that is guaranteed to maintain the state of the system. In these cases the dialogue manager process cannot maintain the state. Thus the process (Fi.g 2b) has to be able to read the previous state S_t along with the input information I_t, and return the new state S_{t+1} and the output O_t before terminating. The new state S_{t+1} is then stored temporarily somewhere else (e.g. in the client browser, in the Hub, in a local database, etc.), before it is sent to a new dialogue manager process that is handling the next interaction.

In order to have the dialogue manager act as a stateless server, one has to manage the bookkeeping of the information to send to the external temporary state repository in order to be able to resume the process seamlessly at the next interaction. Practically the script describing the whole strategy has to be resumed from the exact point it was suspended at the previous turn. The DMD interpreter does this automatically. The same script can run, transparently to the designer, in stateful or in stateless mode. When the DMD interpreter runs in stateless mode and executes a

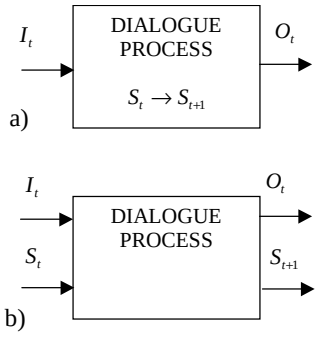

Figure 2: Stateful (a) and stateless (b) dialogue processes

statement causing the suspension of the current flow (e.g. an output statement), it sends the state (including all the necessary information about the current process, e.g. program counter, stack, values of local and global variables, etc.) to the calling process (e.g. the *cgi* script). When the client process invokes the dialogue manager, a new DMD interpreter process is started, the state information is read, and the DMD interpreter reaches the point in the script successive to the last executed statement at the previous turn.
The size of the state for a typical DMD strategy running in stateless mode (e.g. the one managing the travel application described later) is of the order of 3 to 6 Kbytes.

2.4 The Dialogue System Architecture

Fig. 3 shows the overall architecture of the system. A T1 ISDN line is connected to the computer telephony platform (Dialogic based ECTF-compliant CT-media platform). Special software [10] has been included in the telephony platform in order to handle both the speech recognizer and the text-to-speech in an asynchronous manner, thus allowing barge-in. The telephony manager acts as broker. It gets events from the underlying telephony resources, such as recognition results and telephony status (e.g. hang-up, call transferred, etc.), converts them into a template structure, and dispatches them to the DMD interpreter. Conversely, when a template is received from the DMD

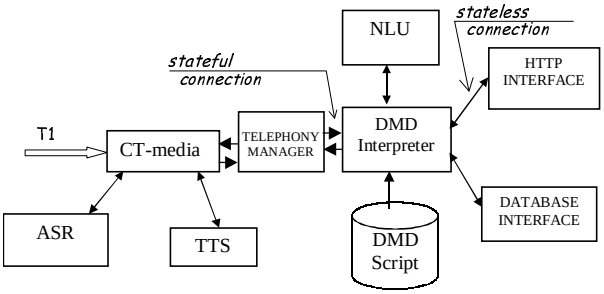

Figure 3: The Dialogue System Architecture

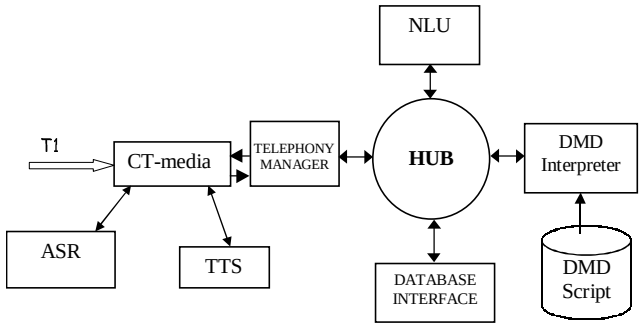

Figure 4: The Communicator compliant architecture

interpreter, the telephony manager looks for information like prompts for TTS, grammars for the recognizer, telephony commands, and sends them to the telephony platform.

The dialogue strategy is coded in the DMD script. The DMD language includes special *Exec* functions that allow for starting and communicating synchronously with external servers. In Fig. 3 we show a natural language understanding server (based on CHRONUS [1]) and a database interface server. The database interface server converts a query semantically expressed by a template into a SQL query, sends it to a relational database server, formats the result as a template, and sends it back to the DMD interpreter.

The DMD interpreter, in this architecture, acts as a server. Each time it is invoked for a new transaction by the telephony manager, it forks a new process that will be active for the whole duration of the dialogue. The same server can be invoked in stateless mode by including the proper control sequence in the initial state template. This is especially useful for text based web interactions. In that case the http interface (a *cgi* script running on a web server) invokes a new DMD process that is forked by the DMD interpreter. The output state is then sent and stored by the browser. At the next turn a new DMD process is invoked with the previous state from the browser.

In this architecture, all the asynchronous events are managed by the telephony platform. The dialogue manager and the other processes it invokes (e.g. natural language understanding) are invoked in a synchronous manner, assuming their processing time is negligible. If the database server delay becomes significant, it should be invoked asynchronously by the telephony platform directly, thus allowing detection of user events (barge-in) while data is accessed.

An alternative architecture, compliant with the DARPA Communicator[9] project, can be obtained with minor changes in the components, as shown in Fig. 4. The Hub is an asynchronous router that receives/sends messages from/to the servers in the form of key-value pairs. The routing function of the Hub is scriptable. In this case the DMD interpreter is always working in stateless mode. When it receives a message (converted into a template format), it processes it according to the DMD script and sends the result back to the Hub, including the current state, so that at the next iteration the script is resumed from the very same point it was left.

3. THE TRAVEL SYSTEM

The travel dialogue[1] system consists of a planning agent for making flight, car, and hotel reservation for multi-leg trips. A typical interaction is shown in Table 1.

3.1 The Strategy

For an effective interaction with the user we adopted a mixed initiative dialogue strategy where the system guides the user by asking questions about specific attributes of the task (e.g. origin, destination, etc.), but at the same time allows for the user to provide more information than the one requested, i.e. the user can provide additional information or correct previously supplied attributes.

Mixed initiative strategies differ form the traditional prompt based or system initiative strategy adopted in IVR and simple voice applications.

A prompt based dialogue strategy can be represented as a tree where each node corresponds to a system prompt, the root corresponding to the initial system prompt (e.g. greetings). Each node branches out with each possible semantically different user response. For example a node corresponding to the question *Would you need a hotel in Boston?* has two branches, one that corresponds to a negative answer and leads, for instance, to the prompt *Do you need a car in Boston?*, and one corresponding to a positive answer leading to the prompt *Would you like a downtown or airport hotel?*.

A prompt based dialogue strategy is usually implemented as a program consisting of nested *if/elseif* statements. A prompt-based strategy is suitable for simple system initiative tasks where the number of semantically different answers is limited at each point.

In a mixed initiative dialogue, instead, the number of possible user inputs is virtually unlimited at any time. Implementing a mixed initiative strategy in a prompt-based manner will result in a prohibitively complex program that is hard or impossible to write and debug. Instead we adopted the sequential decision process model (Fig. 1) for implementing the mixed initiative dialogue, resulting in a significant simplification of the program representing the strategy. In this approach the dialogue strategy is

[1] http://www.research.att.com/~roberto/communicator_web/main.html for more information about the travel system.

specified by two functions of the dialogue state as shown in Fig.1, i.e. *NextAction* and *NextState*.

In the travel task the state is represented mainly by an *itinerary* template that keeps relevant information for each leg of the trip. The *NextAction* function examines the itinerary entries corresponding to the current leg and determines which is the appropriate prompt to invoke. For example, if the origin of the flight is not in the itinerary for the current leg, the next action is the question *What is your departure airport?*. When all the necessary attributes to select a flight (i.e. origin, destination, date, and time) are filled in the itinerary for the current leg, but the flight information is not there, the next action chosen by *NextAction* is a database query.

Welcome to AT&T Communicator. Please say your departure airport.
I am leaving from Denver
Leaving from Denver. Where are you going?
Going to San Francisco
Flying to San Francisco. What date do you want to fly?
Hum?
I need to reconfirm the following information. Please answer yes or no. Are you flying to San Francisco?
Yes
Please tell me the date of your flight.
It is August first late in the morning.
This is the closest flight I have found: United Airlines flight 1521 departs Denver at 11 28 AM and arrives San Francisco International at 1 15 PM. Do you confirm this flight?
Yes
Will you need a hotel?
Yes a hotel downtown
Will you need a rental car?
No thanks
Are you going back to Denver?
Yes
What date do you want to fly?

Table 1: Example of interaction with the travel system

The order in which *NextAction* checks for missing information in the itinerary template is fixed by the strategy script, but the actual order of prompts to the results from the information actually provided by the user. For example, if after a prompt asking for departure city the user specifies, in addition, also the destination, the date and the preferred time, the next action of the system will be database retrieval of the appropriate flight. If the user, instead, always provides just the information that was asked by the system, the dialogue proceeds in a system initiative manner through all the prompts in the prescribed order.

The *NextState* function updates the itinerary template with the current information obtained from the user in the context of the previous prompt. For example if the user says San Francisco and the previous question *Where are you flying to?* it will update the field *destination* in the itinerary template. The *NextState* function handles naturally user initiative by filling the appropriate slots in the itinerary template.

Most of mixed initiative dialogue systems are implemented using a sequential decision approach, such as [2][3][4][6][8].

Notice that while the prompt based strategy can be represented by a tree, the sequential decision process based strategy is represented by the loop of Fig. 1. The same dialogue behavior can be obtained with both strategies, with different resulting complexity of the implementation. In fact, unwrapping the loop of a sequential decision process will result in the tree of the prompt based strategy.

4. CONCLUSIONS

We showed how the abstract sequential decision process model of a dialogue system was used both for designing dialogue architecture as well as for implementing a mixed initiative dialogue strategy.

REFERENCES

[1] Pieraccini, R., Levin, E., "A learning approach to natural language understanding, " in *NATO-ASI, New Advances & Trends in Speech Recognition and Coding*, Springer-Verlag, Bubion (Granada), Spain, 1993.

[2] Stallard, D., "The BBN ATIS4 Dialogue System," Proc. *of 1995 ARPA Spoken Language Systems Technology Workshop*, Austin Texas, Jan. 1995.

[3] Abella, A., Brown, M.K., Buntschuh, B.,"Development Principles for Dialogue-Based Interfaces," Proc. Of European Conference on Artificial Intelligence, Budapest, Hungary, 1996.

[4] Lamel L. et al. "The LIMSI RailTel System: Field Trials of a Telephone Service for Rail Travel Information," Speech Communication, 23 pp.67-82, October 1997.

[5] Levin, E., Pieraccini, R., Eckert, W., "Using Markov Decision Process for Learning Dialogue Strategies, " *Proc. ICASSP 98*, Seattle, WA, May 1998

[6] Seneff, S., Hurley, E., Lau, R., Pao, C., Schmid, P., Zue, V., "Galaxy-II: A Reference Architecture for Conversational System Development, " *Proc. ICSLP 98*, Sydney, Australia, November 1998.

[7] Levin, E., Pieraccini, R., Eckert, W., "A Stochastic Model of Human-Machine Interaction for Learning Dialogue Strategies, " to appear in *IEEE Trans. on Speech and Audio Processing*, January 2000.

[8] Pieraccini, R., Levin, E., Eckert, W., "AMICA, the AT&T Mixed Initiative Conversational Architecure," *Proc. of EUROSPEECH 97*, Rhodes, Greece, September 1997,

[9] DARPA Communicator: http://fofoca.mitre.org

[10] G. Di Fabbrizio, C. Kamm , P. Ruscitti, S. Narayanan, B. Buntschuh, A. Abella, J. Hubbell, J. Wright "Extending a Standard-based IP and Computer Telephony Platform to Support Multi-modal Services, " Proc. of *ESCA Tutorial and Research Workshop on INTERACTIVE DIALOGUE IN MULTI-MODAL SYSTEMS*, Kloster Irsee, Germany - June 22 -25, 1999.

[11] Dialogic CT Media: http://www.dialogic.com/products/ctmedia