Spoken Language Dialogue Architectures and Algorihtms

Roberto Pieraccini, Esther Levin, Wieland Eckert

AT&T Labs-Research
180 Park Avenue, Florham Park, NJ 07932, U.S.A
Tél.: (973) 360-8563 - Fax: (973) 360-8092
e-mail: {roberto, esther, eckert}@research.att.com - http://www.research.att.com

ABSTRACT

In this paper we discuss the evolution of spoken language dialogue architectures and algorithms at AT&T Labs-Research. First we introduce user initiated systems that are able to understand user requests and just provide answers. Then we discuss mixed initiative systems that can take the initiative when necessary in order to reach the application goal in the most effective way. Furthermore we introduce one of the most ambitious goals in the field today, namely learning the dialogue strategy.

1. INTRODUCTION

One of the ultimate goals of the speech recognition community is the implementation of machines able to interact with humans in a natural conversational way in order to provide services that would otherwise require human operators or unfriendly menu based systems.

Although research in the field of human-machine spoken dialogue started decades ago, only recently [9] were we able to realize end to end systems. The ability of building such systems can be attributed without any doubts to the recent advances in the related technologies. First, the ability of building high accuracy large vocabulary spontaneous speech recognizers that can work in real time on inexpensive computers. Second, the development of robust language understanding systems that can deal with the problems inherent to spontaneous speech. Third, the recent flourishing of research aimed at the problem of dialogue system architectures, and trying to solve issues such as modularization, porting, reusing, fast prototyping, etc.

In this paper we will discuss some of the evolution of dialogue systems research at AT&T Labs-Research. This effort started with the implementation of language understanding systems suited for spontaneous speech that could be regarded as a primitive form of dialogue system that allow only user-initiated interactions. Then a more complex architecture for mixed-initiative dialogue was investigated and implemented. Finally we proposed a mathematical formalisation of the human-machine dialogue that will allow for learning of dialogue strategies.

2. LANGUAGE UNDERSTANDING AND USER-INITIATED DIALOGUE

A user-initiated dialogue system consists of a machine that responds to user's questions with the best possible answer given the context of the whole session. The evolution of dialogue is then a succession of user's questions and system's answers. User initiated dialogue can be generally implemented through a cascade architecture, an example of which (for a database retrieval application) is shown in Figure 1. Here is a brief description of the process:

1. The **speech recognizer** transcribes input speech into a sequence of words.
2. The **lexical analyzer** attaches tags to words (or short phrases) that are semantically meaningful for the application (e.g. numbers, proper names, modifiers, etc.). Since some words can have multiple interpretations within the same application (e.g. the proper noun *New York* can be interpreted either as a *state* or as a *city*), the output is a lattice of word hypotheses (rather than a string)
3. The **conceptual decoder** finds contiguous sequences of words hypotheses in the lattice and assigns them labels belonging to a finite set of concepts that are relevant for the application (e.g. origin, destination, date, etc. in the case of a travel reservation application.). The output of this module is called *conceptual segmentation.*
4. The **template generator** constructs a symbolic representation of the conceptual segments and represents it as a set of key/value pairs. This is the *local meaning representation*, that is the representation of the meaning of the current sentence independently from the previous user sentences in the session.
5. The **interpreter** builds a more accurate representation of the meaning taking into account interactions between two or more concepts in the same sentence, the context set by the previous sentences, and specific interpretation rules for the application. The output of this module, the *contextual meaning representation*, is again a set of key/value pairs.
6. The **database interface** translates the contextual semantic representation into a database query. The

resulting data is then transformed into the desired format (e.g. a table, a list, etc.) and sent to the user.

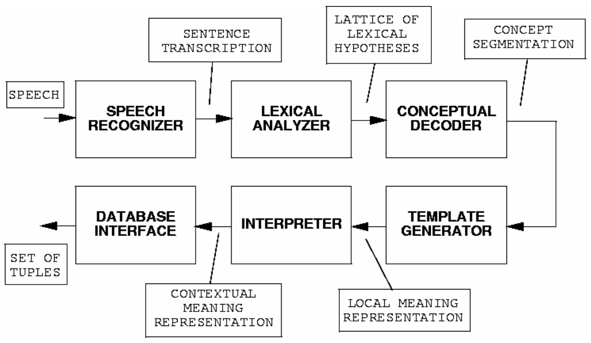


Figure 1: The CHRONUS spoken language understanding system.

In the rest of this section we will give a short description of each module, with reference to the CHRONUS system [5] that was used within the ARPA ATIS [2] project.

2.1 Speech Recognizer

We will not describe the speech recognizer in this context, except saying that is a HMM sub-word based recognizer [6] using a variable *n* n-gram stochastic language model [7] that produces the first best sentence hypothesis in the for of a plain string of text.

2.2 Lexical Analyzer

The function of the lexical analyzer is that of creating a lattice of word hypotheses out of a string of words (i.e. the transcription of speech into words generated by the speech recognizer [5].

The different hypotheses in the lattice correspond to possible interpretations of words and/or phrases according to predefined semantic categories. Most of the categories correspond to classes of attributes of the database (e.g. cities, dates, etc.) words that we decided to consider as synonyms for later stages of the process. For instance, the distinction between singular and plural forms of words as well as different verb inflections is irrelevant for many tasks, so they are collapsed in the same category. This both reduces the effective size of the lexicon and enhances the robustness of the stochastic conceptual representation [1], especially in presence of recognition errors (speech recognizers are likely to confuse among different inflections of the same word). Numbers and acronyms are parsed in all the possible ways, each parse constituting a separate lattice hypothesis.

2.3 Conceptual Decoder

The idea behind the conceptual decoder is that of treating a sentence as a sequence of phrases corresponding to units of meaning called concepts. A Markovian process models the sequence of concepts (a first order process in the current implementation).

The sequence of words that forms a phrase related to a given concept is also modeled by a Markovian process represented by a concept conditional n-gram language model.

In the current implementation the concept conditional language models are bi-gram back-off networks [6]. The overall model is called conceptual HMM [1][3].

The function of the conceptual decoder is that of segmenting a sentence into phrases and assigning each phrase to the correspondent concept. This translates into finding the most likely sequence of states in the conceptual HMM given the lattice produced by the lexical analyzer and it is implemented as a finite state automata intersection operation as explained in.

There are two problems in designing a conceptual decoder for a given application: the choice of the conceptual units and the training of the conceptual HMM. The choice of the conceptual units generally requires a good knowledge of the task. Most of the units generally correspond to entities of the database.

Training a conceptual HMM implies providing a considerable amount of examples of conceptual segmentations and running a Viterbi training algorithm [1] in order to estimate the parameters of the model. Of course the most cumbersome part is providing examples of segmentations, namely segmenting by hand thousands of sentences. This can be enormously alleviated using a training loop procedure [3] that uses an automatic evaluation criterion to detect possibly wrong segmentations in a segmented corpus.

2.4 Template Generator

The template generator produces a representation of the sentence semantics in the form of a template, i.e. an unordered set of keyword/value pairs (called *tokens*) [3] starting from the segmentation produced by the conceptual decoder.

Most tokens correspond to attributes of the database and their values. For instance the token DEPARTURE_CITY:SSFO corresponds to assigning the value SSFO (i.e San Francisco) to the database attribute departure_city.

There are tokens that act as modifiers and are recognized either by the interpreter or by the database interface. Examples of this are the token QUESTION (whose value YES/NO instructs the database interface to produce a yes/no answer), or the token DEPARTURE_PERIOD (whose value modifies any DEPARTURE_TIME token in the same template to AM or PM accordingly).

The requested information is specified by SUBJECT tokens whose values have a direct correspondence to the attributes of the database.

2.5 Interpreter

The function of the interpreter consists in resolving ambiguities and providing missing information in the current template.

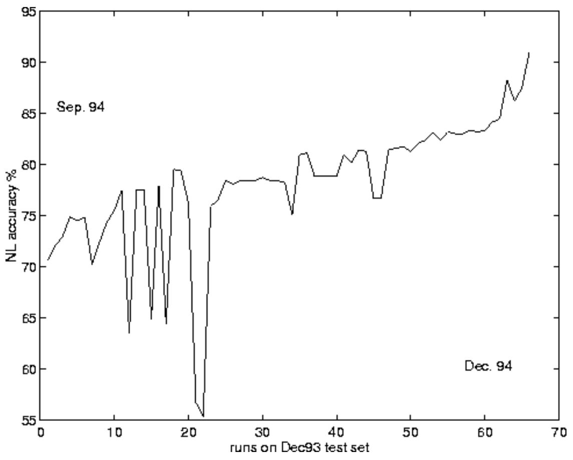


Figure 2: Incremental improvement of the ATIS system

In general the interpreter merges the current template with a *context* template according to a set of masking rules. The resulting template is used for further processing and stored as the new *context* template.

One example of general masking rule states that tokens in the most recent template mask tokens in the context template with the same key. One example of a specific masking rule states that a YEAR token in the most recent template masks DATE tokens in the context template (i.e. the specification of a different year makes the previous date invalid).

The interpretation rules are generally written by hand and special care should be taken for them to be consistent. A corpus and an evaluation procedure are very useful for the development of this module, since they allow to set up an automatic way of assessing the effectiveness and the consistency of each newly introduced rule. In the development of the ATIS interpreter every time new rules were introduced the system was evaluated on a corpus of about 5,000 sentences (this test was completely automated and required only a few hours of CPU time).

Figure 2 shows the percentage of correct sentences in almost 70 experimental runs during the three months in which the ATIS interpreter was developed.

2.6 Database Interface

The database interface takes the semantic representation in the form of a template and extracts the requested information from a relational database. In general it is necessary to provide some specific piece of software that will convert the template representation into the query language of the database (e.g. SQL, web-based forms, etc.). Other more general solutions can be adopted in the case the data is local and represented in a relational form [13].

1.1 System Performance

The performance of the system based on the CHRONUS approach is shown in Figure 3 where the official figures for the three standard tasks in the ATIS program are reported [5]. The scores for the speech recognition task, the text understanding task and the speech understanding task were among the best across all sites participating to the evaluation.

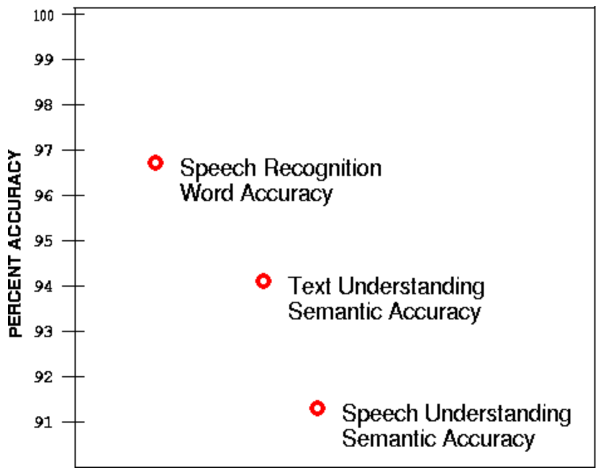


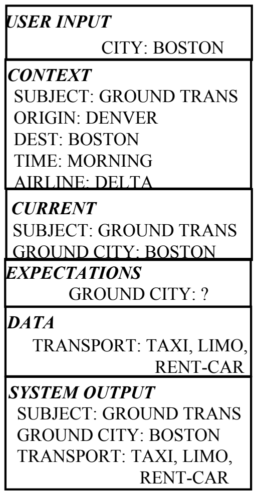
Figure 3: Performance of the AT&T system in the 1994 ARPA ATIS official evaluation.

3. MIXED INITIATIVE DIALOGUE ARCHITECTURE

A user initiated system has many obvious limitations for complex applications like database retrieval. For instance when an initial question like *I need some information about flights to New York* is asked, a system that does not have the possibility of asking questions (such as a user initiated system) will respond showing all the flights to New York from any other city in the database. Such a long response may not be appropriate for the kind of output medium (e.g. an audio response). Moreover the cost of accessing a remote database with such an unrestricted question and the time for transferring the results can be excessive

M*ixed initiative* systems constitute the natural evolution of the user initiated systems. A mixed initiative system shares with the user the right of asking questions as well as giving answers and changing the course of the conversation when necessary. Table 2 shows an example of mixed initiative interaction.

Mixed initiative can hardly be implemented by simple cascade architectures. Moreover it would be desirable to have a *general* architecture that would easily fit different



USER INPUT
CITY: BOSTON

CONTEXT
SUBJECT: GROUND TRANS
ORIGIN: DENVER
DEST: BOSTON
TIME: MORNING
AIRLINE: DELTA

CURRENT
SUBJECT: GROUND TRANS
GROUND CITY: BOSTON

EXPECTATIONS
GROUND CITY: ?

DATA
TRANSPORT: TAXI, LIMO, RENT-CAR

SYSTEM OUTPUT
SUBJECT: GROUND TRANS
GROUND CITY: BOSTON
TRANSPORT: TAXI, LIMO, RENT-CAR

Figure 4: Representation of the dialogue state

applications, and would allow for a rapid development of new systems.

Failure to find such a general architecture in the past derived generally from the lack of separation among the different levels of competence that intervene during the dialogue activity.

AMICA [13], the AT&T Mixed Initiative Conversational Architecture, is based on the identification of a set of general, logically motivated functions, called *dialogue actions*. We think that for certain classes of dialogues there exists a finite (and small) number of such actions that can be implemented in a general way and parameterized in order to be used for different applications.

The following is a basic set of dialogue actions:

- **Understand**: transforms the user natural language input into a semantic representation (it corresponds to the speech recognizer, lexical analyzer, conceptual decoder, and template generator of section 0).
- **Verbalize**: transforms a semantic representation into a natural language output. The action of verbalizing data retrieved from the database is called **verbalize data**.
- **Context Tracking**: consists in the interpretation of the current user input in terms of the history of the interaction. It includes both *dialogue interpretation*, namely the ability of dealing with expectations set by the machine itself (e.g. disambiguation on the basis of previous questions asked by the machine), and *discourse interpretation*, namely the ability of taking into account the context set by the user during the course of the whole interaction (corresponding to the interpreter of section 0). In both cases it is necessary to be able to recognize ambiguities.
- **Retrieve Data**: consists in the assembly and submission of a query to a local or remote database according to the current information gathered by the system.
- **Constrain:** consists generating a request for additional constraints based on the current topic and information available. particular topic.
- **Relax:** consists in relaxing attributes of the query in order to perform approximate queries to the database.

3.1 Dialogue Control

For explaining how the AMICA architecture works it is necessary to introduce the concept of *system state*. The system state is the collection of all the pieces of information that characterize the whole dialogue system at a certain point in time and will allow to perform the next step of the interaction (to that respect the dialogue system can be considered memoryless).

The system state is represented by frame like (i.e. key/value pairs) data structure similar to the one shown in Figure 3.

The *dialogue strategy* is a mechanism that, given the current state of the system, selects one of the possible actions. As an effect of the execution of the selected action, the system state changes, then a new action is selected by the strategy, etc.

First of all we have to realize that, in any realistic application, the number of possible system states in extremely large or infinite. Hence it is not possible to proceed with the enumeration of all the states and assign each one of the to a particular action.

Rather the selection of the proper action to execute given the current state of the system is based on conditions on the state. In the most general case the control can be implemented as a finite state recursive transition network whose nodes are associated with actions and whose (ordered) transitions are associated with conditions. When the control is in a generic node, the corresponding action is executed (as a result the system state is modified). Hence the conditions associated to the transitions from that node are evaluated in a prescribed order. The first condition that is verified leads to the next action, and the process is repeated again.

An example of simple strategy for mixed initiative dialogue is shown in Fig. 4 (*Context Tracking* has been included in the *Understand* action in this example).

The control starts with the *verbalization* of a greeting. U*nderstand* gets user's input (interpreted by the *context tracking* function). If the current user request is underspecified, *Constrain (1)* will generate a request for additional constraints that will be appropriately verbalized. Otherwise *Retrieve* will access the database with the current request. If too much data is returned by the database (*NDATA > t*), then *Constrain (2)* will generate a request for additional constraints that will be verbalized. If the database returns no data (NDATA == 0), *Relax* will relax constraints until some data is retrieved. If a reasonable amount of data is retrieved (0< NDATA <= t), it will be verbalized and a new request from the user will be collected.

1	VERBALIZE	This is the AT&T Chronus Air Travel Information System, how may I help you?
2	UNDERSTAND	**I want a direct flight from Denver to Boston between eight and nine thirty tomorrow morning**
3	RETRIEVE RELAX RETRIEVE VERBALIZE DATA	There are no flights from Denver to Boston leaving between 8 and 9:30 in the morning. Instead there are flights leaving at 10:39 and 11:05. Do you have any preference?
4	UNDERSTAND	**Give me a delta flight**
5	UNDERSTAND RETRIEVE VERBALIZE	Flight DL296 leaves at 10:39 am and arrives at 3:15 pm. Other questions?
6	UNDERSTAND	**I need ground transportation**
7	CONSTRAIN VERBALIZE	Where, Denver or Boston?
8	UNDERSTAND	**Boston**
9	RETRIEVE VERBALIZE DATA	The following ground transportation is available in Boston: taxi, limousine, rental car.
10	UNDERSTAND	**Please list all the flights with Continental**
11	RETRIEVE CONSTRAIN VERBALIZE	There are too many flights that match your request. I need more information. At what time you want to leave?

Table 1: Example of dialogue with the AT&T Arline Information System

This simple strategy is able to generate a reasonable behavior for database retrieval applications. The behavior of the system during a session evolves through three interaction modes, corresponding to loops in the strategy of Figure 5. The first loop, that we call *minimal information* (Understand – Constrain (1) – Verbalize) will gather enough information from the user in order to perform a reasonable database query. In fact, the absence of this behavior will force to retrieve data even when the user did not give the minimal required information. An example from the ATIS domain is the user asking "I would like information about flights to New York", and the system requesting from the database *all the flights* to New York from any airport, a useless expensive query!

The second loop, that we call *constraining*, (Understand – Constrain (1) - Retrieve – Constrain (2) - Verbalize) will attempt to reduce the amount of data *presented* to the user. Depending on the channel, there is a maximum amount of data the user will be able to assimilate. Typically two or three pieces of information (tuples) with an audio channel, 10 or more with a small display, etc. If more data is retrieved, other questions need to be asked in order to trim it down to a reasonable quantity.

Finally the third loop, the *relaxation* loop (Understand - Constrain (1) - Retrieve – Relax – Retrieve …. Relax – Retrieve – Verbalize) attempts to give the user some data in presence of over-constrained request (e.g. *I would a flight that leaves (exactly) at 7 a.m.*).

An example of a fragment of dialogue drawn from the Airline Information System and using the strategy of Figure 4 is shown in Table 1.

3.2 Portability Issues

The process of developing a dialogue system for a new application is greatly eased by the functional independence of the various modules that compose the system architecture. Moreover all the actions described so far can be implemented in a *table driven* or *programmable* fashion.

We will give here some examples of the action programmability in order to stress the application independence of the system.

3.2.1 Verbalize

The verbalization (or sentence generation) process is implemented by an interpreter that accepts sentence

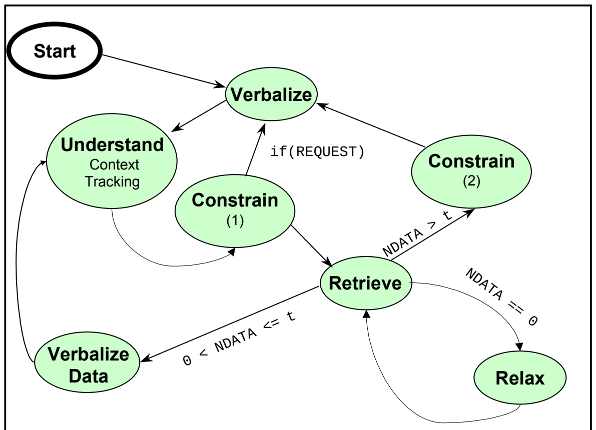


Figure 5: Example of mixed initiative strategy

archetypes with conditional clauses. An example of a sentence archetype is the following:

```
if(NDATA > 1) {"There are "}
          else {"There is "}
NDATA if(NDATA > 1) { "flight " }
                   else {"flights"}
  "from " DEPARTURE_AIRPORT
       "to " ARRIVAL_AIRPORT ".";
```

This archetype can generate sentences like:

There are 5 flights from Newark to Denver.
There is 1 flight from San Diego to Boston.

For the verbalization of relational database tuples the process relies on the semantic description of the database augmented by grammatical information. For instance verbs and prepositions are attached to pairs of database attributes, like:

```
FLIGHT_ID  flies to DEPARTURE_AIRPORT
FLIGHT_ID leaves from ARRIVAL_AIRPORT
FLIGHT_ID is served by AIRLINE
FLIGHT_ID uses a AIRCRAFT
```

These pairs of predicates are organized into a finite state network that is then used for building sentences with an arbitrary number of attributes, like

Flight UA706 leaves from Denver, flies to Newark, is served by United, and uses a Boeing 747.

3.2.2 Constrain

The constrain action, in the way it was implemented in this system, requires the specification of logical expression whose truth guarantees that a set of constraints for a given topic is available in the current state. Still with reference to an airline application, the following is a valid set of constraints for the topic FLIGHT:

```
if(TOPIC: FLIGHT) ask
  (AIRLINE && FLIGHT_NUMBER) ||
(DEPARTURE_AIRPORT &&
           ARRIVAL_AIRPORT);
```

The constrain algorithm matches this expression with the current state (each symbol is true if the corresponding attribute is present in the state). If the expression is false, another logical expression is computed whose truth will make true the original expression. For instance, if the state includes the following information:

```
AIRLINE : DL
DEPARTURE_AIRPORT: EWR
```

The constrain algorithm will generate the following expression:

```
MISSING: FLIGHT_NUMBER ||
ARRIVAL_AIRPORT
```

Corresponding to the question:

Please specify your flight number or arrival airport

3.2.3 Relax

The relaxation action, in its simpler implementation, requires a list of domain attributes order according to their relative importance. For instance, in the airline information task, the list can be: origin/destination, date, time, type of aircraft, meal. Attributes are relaxed in the order until a non empty set of tuples is found.

In certain application the relaxation procedure is more complex, and special functions have to be appositely designed. For instance, in a movie locator application, one can be interested in relaxing the *city* attribute, hence finding which one is the theater *closer* to a given city that shows a certain film. This requires a specific algorithm that computes the distance of theaters from cities (the may not be provided by the database) and searches for the closest one.

4. FORMALIZATION AND LEARNING OF MIXED INITIATIVE DIALOGUE SYSTEMS

As we saw in the previous section the strategy determines the behavior of the system, and it is generally built by hand based on a good knowledge of the domain. For instance a reasonable strategy for the airline domain is to request the origin and the destination of the flight and the departure date at the beginning of the transaction, because we know that those are the most distinctive questions that will greatly restrict the search in the database. But when faced with a new domain the choice cannot be that obvious. Moreover the criterion on which we base our reasonable strategy (e.g. length of the interaction and database retrieval in the airline example) can be different from task to task. It would be highly desirable to have a mathematical formalization of the strategy design process, and to derive procedures for automating it. This is the subject of the following sections.

4.1 Dialogue Systems as Markov Decision Processes

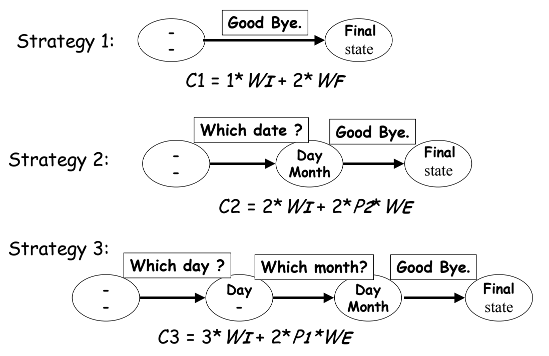


Figure 6: Three different strategies

For explaining the formalization we use for describing the dialogue process, we will use a very simple tutorial example. We will assume that the goal of our dialogue system is getting a date from a user through the *shortest* possible interaction.

We will formalize a dialogue system by describing it in terms of a **state space**, **action set**, and **strategy**.

In our simple tutorial example, the **state** of the system includes only two entries: the day and the month, whose values can be either empty, or filled through interaction with the user. The total number of states is 411, including one empty initial state, 12 states for which the month is filled and the day is not, 31 states in which the day is filled, but not the month, 366 states with complete dates, and a special final empty state.

The action set include only four possible actions:

1. Asking **which day**.
2. Asking **which month**.
3. Asking **which date**. This is a more open-ended question and will compel the user to answer with a day and a month in the same sentence.
4. A **final action**, closing the dialogue and submitting the information.

In actions 1, 2 and 3, the system asks the appropriate question, and activates a speech recognition system in order to obtain the user's answer. We assume that the probability of error of the speech recognizer depends on the kind of question. In particular questions 1 and 2 will both show a different word error probability than question 3 (being the user's answer longer and more articulated).

Once the system is in a given state (for instance the initial empty state) and an action is executed (for instance the system asks *which month*), the system goes into another state (in this case, depending on the user's answer, one of 12 possible states, assuming the user is compliant). Since the new system state resulting from the execution of an action depends on external causes (for instance the user's response), we model it by transition probabilities $P_T(s_{t+1} = s' \mid s_t = s, a_t = a)$.

A **dialogue session** corresponds to a path in the state space starting at the initial state and ending at a final state, the **dialogue strategy** specifies, for each state reached, what is the next action to be invoked.

At this point we introduce an objective measurement of the quality of a dialogue session based on a cost function:

$$C = \sum C_i , \qquad (1)$$

that is the sum of individual costs incurring during the interaction. Those individual costs

Ci measure the effectiveness of the interaction (e.g. its length), the distance from the application goal (e.g. whether the system provided the user the requested service), and other costs, like database access costs. Furthermore it has been shown [13] that also subjective measurements like *user satisfaction* can be modeled as a linear combination of costs, and could therefore used as one of the terms in equation (1).

Now we can define that an optimal dialogue strategy is the one that minimizes the expected value of cost *C*.

For our tutorial example, where the goal of the system is to obtain the correct day and month values through the shortest possible interactions, the objective function includes three terms:

$$< C >= W_I \cdot < N > + W_E < E > + W_{UF} < UF >$$

Where <*N*> is the expected duration of the dialogue; <*E*> is the expected number of errors for the filled slots (ranging from 0 to 2); and <*UF*> is the expected number if unfilled slots. The optimal strategy, i.e. the one that minimizes <*C*>*,* will be a trade off between duration and errors (in fact asking question 3 rather than 1 and 2 will produce a shorter but more error prone dialogue), and will depend on the values of the error probabilities.

In order to reflect this objective function in our dialogue model, we associate a cost *c* to the action *a* selected while in a state *s* (in general we can say that action costs are described by conditional distributions $P_C(c_t = c \mid s_t = s, a_t = a)$).

The cost incurred with any of the first three actions in day-and–month dialogue system is $W_I + W_E \cdot E$. If we assume that the concept error rate for recognition of month or day for questions 1 and 2 is p_1, for question 3 is p_2, with $p_2 > p_1$, then the expected cost accumulated when actions 1 or 2 are taken is $W_I + W_E \cdot p_1$, while for question 3 is $W_I + 2 \cdot W_E \cdot p_2$. The cost of action 4 (closing the dialogue and submitting the obtained date) depends deterministically on the state in which this action is taken and is $W_I + 2 \cdot W_{UF}$ for an initial state, $W_I + W_{UF}$ for states in which either the day or month is unfilled, and W_I for states in which both values are filled.

Of course, different strategies for the same system result in different expected session costs. Figure 5 shows three different strategies and their associated expected costs. For example strategy 1 (where the system does not even engage in dialogue, closing the dialogue as the first action) is optimal when the recognition error rate is too high, i.e. $p_1 > (W_{UF} - W_I)/W_E$.

In strategy 2 the system opens the dialogue by asking the open ended question 3, fills out the day and the month slots with the values recognized from the user response, and closes the session. In strategy 3, the system first fills up the day and then the month by engaging in actions 1 and 2, and then closes the session. Strategy 3 is optimal when the difference in error rates justifies a longer interaction*:* i.e. when $p_2 - p_1 > W_I/(2 \cdot W_E)$.

The model we just described is known as Markov decision process[8], or MDP, and is in general described by the quadruple: state space, action set, transition probabilities, and cost distributions.

4.2 Automatic Design of Man-Machine Dialogue Systems: the Reinforcement Learning Approach.

Stating the problem of man-machine dialogue design as an optimization problem provides the following potential advantages:

Objective evaluation: It is possible now to assess different strategies for the same system just by comparing their expected cost. It is also possible to compare different systems that share the same objective function.

Automatic design: Since the problem of strategy design is cast as optimization problem, it is possible to devise methods for performing this optimization automatically. In the simple example of the previous section it was possible to derive the optimal strategy analytically. This is not true in general with realistic applications. However there are learning techniques, such as reinforcement learning [8], that allow finding the optimal strategy in an MDP from interactions.

4.2.1 An experiment with reinforcement learning.

Rather than explaining here the details of the reinforcement learning algorithm, we will discuss the results of an experiment devoted at automatically finding a strategy for a simplified version of the airline information task.

We restricted the possible actions of the system to the following:

Greet the user with an open ended question (i.e. How can I help you?);
Ask the user to provide information about a specific attribute of the task (e.g. origin, destination, airline, departure time, etc.);
Retrieve data from the database according to the current user request (this action does not involve interaction with the user);
Output data to the user;
Ask the user to **release a constrain**;
Close the dialogue.

In order to simplify the learning task we chose a very simple state description. The state included three fields: the USER field represents the user request (only the keywords), the DATA field includes just the number of data tuples retrieved from the database, and the SYSTEM field records whether the actions **retrieve** or **output** have been already used .

We decided to use the following cumulated cost:

$$C = W_1 \cdot N + W_2 \cdot R + W_3 P + W_4 U$$

where N is the number of interactions, R is the cost of data retrieval (linear with the number of tuples retrieved), P is cost of data presentation (zero for a acceptable number of tuples and rapidly increasing with the number of tuples thereafter), and U is the user dissatisfaction cost that penalizes, with a fixed cost, dialogues that fail to provide any flight information to the user.

The system starts with no knowledge of the task, i.e., each action had the same probability of being selected by the system at any step. A typical dialogue with an untrained system will look like the following:

S	RELEASE AIRLINE	*Do you want to choose another airline?*
U		What?
S	RETRIEVE	(6,893 flights retrieved)
S	CONSTRAIN DEPARTURE TIME	*When do you want to leave?*
U		Uh? I want to go to Boston
S	OUTPUT DATA	*There are 6,893 flights ....*
U		I don't understand, can you please show me the flights from San Francisco to Boston?
S	CLOSE DIALOGUE	*Thank you for using AT&T.*

Of course the total cost for this dialogue is quite high, especially due to the high cost of retrieval (all the database was retrieved here) and data output (6893 flights).

For learning the optimal strategy we used a Monte Carlo style reinforcement learning algorithm with exploring starts, as explained in [17]. With reinforcement learning it is not possible to train a system using a fixed corpus of interaction, since the system is learning while interacting, and the strategy is changing from iteration to iteration. Also it is not realistic to use real users for the learning stage that can be very long (thousands of interactions) and annoying (see the previous dialogue). So we decided to use a stochastic user model [15] designed to simulate the interaction. The user model includes probability distributions over user's dimensions such as compliance in answering specific questions and number and ype of unsolicited constraints provided in open ended questions. Those parameters were estimated on the ATIS corpus.

With the user model interacting with the reinforcement learning algorithm for thousands of interactions the system was able to learn the following optimal strategy:

start the dialogue by **greeting** (the probability of getting more information from the user is higher with greeting rather than a specific question),

ask **constraining** questions until the origin, destination, and airline are specified,

retrieve data from the database.

if the resulting data set is empty (because the query was over-constrained), **release** the airline constraint and **retrieve** again.

if there are too many flights in the data set, ask to **constrain** the departure time and retrieve again.

if at any point during the dialogue the retrieved data set has a reasonable number of flights, the data is presented to the **output** and the dialogue is **closed**.

An example of dialogue performed with a trained system is as follows:

S	GREETING	*This is the AT&T flight information system, how may I help you?*
U		I want to go to Boston
S	CONSTRAIN ORIGIN	*What is your departure airport?*
U		San Francisco.
S	CONSTRAIN AIRLINE	*Which airline?*
U		Delta.
S	RETRIEVE	(30 flights retrieved)
S	CONSTRAIN DEPARTURE TIME	*When do you want to leave?*
U		Late in the afternoon.
S	RETRIEVE	(3 flights retrieved)
S	OUTPUT DATA	*Flight ... leaves at ..., flight ... leaves ...*
U		Thanks.
S	CLOSE DIALOGUE	*Thank you for using AT&T.*

This strategy is the same as the one that was designed by hand and it I shown in Figure 4. This experiments demonstrates that it is possible build systems that are able to learn (and eventually to adapt) automatically what is the best dialogue strategy based only on the enumeration of the possible actions and on the definition of sensible cost functions.

Summary

We discuss in this paper the evolution of spoken language dialogue systems starting from the user-initiated model, where the user asks questions and the system provides only the most reasonable answer in the context of the previous interaction. The following step in the evolution of dialogue system is the mixed initiative. In this model, the system can take the initiative at any time during the conversation in order, for instance, to gather additional information from the user for a more effective achievement of the goal. We described a general architecture for implementing mixed initiative system, based on the concepts of system state, dialogue actions, and strategy. This architecture naturally leads to a formal definition of a dialogue system in terms of a Markov decision process, and to the ambitious goal of learning the dialogue strategy while interacting. We show that learning is possible even in complex applications, and that reinforcement learning algorithm can find reasonable strategies that are optimal for a given cost function.

REFERENCES

[1] Pieraccini, R., Levin, E., "Stochastic Representation of Semantic Structure for Speech Understanding," Speech *Communication*, Vol.11 pp. 283-288, 1992.

[2] MADCOW, "Multi-Site Data Collection for a Spoken Language Corpus," *Proc. Of Fifth Darpa Workshop on Speech and Natural Language*, Harriman, NY, Feb 1992.

[3] Pieraccini, R., Levin, E., "A learning approach to natural language understanding," in *Speech Recognition and Coding, New Advances and Trends*, NATO ASI Series, Springer, 1993.

[4] Pereira, F., Riley, M. D., Sproat, R., "Weighted rational transductions and their application to human language processing," ARPA Human Language Technology Workshop, Princeton, NJ, March 1994.

[5] Levin, E., Pieraccini, R. "CHRONUS, The Next Generation," Proc. of *1995 ARPA Spoken Language Systems Technology Workshop*, Austin Texas, Jan. 1995.

[6] Bocchieri, E.L., Riccardi, G. Anantharaman, J., "The 1994 AT&T ATIS CHRONUS Recognizer," Proc. of *1995 ARPA Spoken Language Systems Technology Workshop*, Austin Texas, Jan. 1995.

[7] Riccardi, G., Bocchieri, E., Pieraccini, R., "Non Deterministic Stochastic Language Models for Speech Recognition," Proc. of ICASSP 95.

[8] Kaelbling, L. P., Littman, M. L., Moore, A. W., "Reinforcement Learning: A Survey," in *Journal of Artificial Intelligence Research*, No. 4, pp. 237-285, May 1996.

[9] Sadek, M., D., et. al. "Effective Human-Computer Cooperative Spoken Dialogue: the AGS Demonstrator," in *Proc. of ICSLP 96*, Philadelphia, PA, Oct. 96.

[10] Gorin, A. L., Parker, B. A., Sachs, R. M. and Wilpon, J. G., ``How may I help you?,'' Proc. Interactive Voice Technology for Telecommunications Applications (IVTTA), Oct. 1996, pp. 57-60.

[11] Kamm, C., Narayanan, S., Ritenour, R. and Dutton, D., "Evaluating spoken dialog systems for telecommunication services," *Proc. Eurospeech97*, Greece, Sept. 1997, pp. 2203-2206.

[12] Walker, M., A., Hindle, D., Fromer, J., Di Fabbrizio, G., Mestel, C., "Evaluating Competing Agent Strategies fro a Voice Email Agent," *Proc. Eurospeech97*, Greece, Sept. 1997, pp. 2219-2222.

[13] Pieraccini, R., Levin, E. and Eckert, W., "AMICA: the AT&T Mixed Initiative Conversational Architecture," *Proc. Eurospeech 97*, Rhodes, Greece, Sept. 1997, pp 1875-1878

[14] Walker, M. A., Littman, D. J., Kamm, C. A., Abella, A., "PARADISE: A Framework for Evaluation of Spoken Dialogue Agents," in Proc. *35-th Annual Meeting of the Association for Computational Linguistics,* Madrid, Spain, 1997.

[15] Eckert, W., Levin, E. and Pieraccini, R., "User modeling for spoken dialogue system," ASRU '97, Proc. of *1997 IEEE Workshop on Automatic Speech Recognition and Understanding*, Santa Barbara, CA, Dec.1997.

[16] Gorin, A. L., Riccardi, G. and Wright, J. H., ``How may I help you?,'' Speech Communication, To appear.

[17] R. S. Sutton, A. G. Barto, "Reinforcement Learning, an Introduction," MIT Press, 1998.