

THE CU COMMUNICATOR SYSTEM¹

Wayne Ward and Bryan Pellom

Center for Spoken Language Research
University of Colorado, Boulder
Boulder, Colorado 80309-0594, USA
<http://cslr.colorado.edu>

ABSTRACT

The CU Communicator system is our initial testbed for research leading to advanced dialog systems enabling robust and graceful human computer interaction. It is a DARPA hub compliant system for the DARPA Communicator task, and was demonstrated at the DARPA workshop in June 1999. Robustness and portability of spoken dialog systems are two of the issues we attempt to address in the project. We use robust parsing and dialog control strategies to be as flexible as possible to user variance. In order to make the systems easier to develop, we have adopted a largely declarative representation where the bulk of the domain specific information is provided in external files.

1. INTRODUCTION

In April 1999, the University of Colorado speech group began development of the CU Communicator system, a Hub-compliant implementation of the DARPA Communicator task[1][2]. The system combines continuous speech recognition, natural language understanding and flexible dialog control to enable natural conversational interaction by telephone callers to access information about airline flights, hotels and rental cars. In June 1999, the system was fully functional, and was demonstrated at the June 1999 DARPA Communicator meeting.

This system connects live over the web to get real up-to-date air travel, hotel and rental car information. Users call the system on the telephone and make travel plans. We are using the system as a testbed for conversational system technology development. For this system, we developed a new Dialog Manager using an event driven strategy. Our natural language and dialog specification for the task is very much declarative.

Our approach to robustness in natural language understanding is to use robust parsing strategies and event driven dialog strategies. These strategies stress semantic content and coherence over syntactic form. We have extended the Phoenix system as the basis for our robust parsing. This system uses a hierarchical frame parse representation. Our "event driven" dialog manager does not use an explicit script. The developer creates a set of hierarchical forms, representing the information that the system and user interact about. Prompts are associated

with fields in the forms. The dialog manager decides what to do next from the current system context, not from a script.

We also attempt to address issues of authoring and portability. Building a spoken dialog system is labor intensive and requires a great deal of expertise. A major focus of our research is the development of tools to facilitate rapid creation of spoken dialog applications by non-expert developers. Our initial steps have been:

- Declarative representation – To the extent possible, we are representing information in external files rather than having the developer write code.
- Libraries – We are creating a library of common grammars that would be applicable to many tasks (times, dates, etc).

2. SYSTEM OVERVIEW

The CU Communicator uses a DARPA Hub compliant architecture, which means that the system is composed of a number of servers that interact with each other through the DARPA Hub. The system is composed of the Hub and seven servers;

1. Audio – receives signals from telephone and sends to recognizer. Sends synthesized speech to telephone
2. Speech Recognition – Takes signals from audio server and produces a word lattice
3. Natural Language Processing – Takes word lattice from recognizer and produces the best interpretation
4. Dialog Management – Resolves ambiguities in the interpretation, estimates confidence in the extracted information, clarifies with user if required, integrates with current dialog context, builds database queries (SQL), sends data to NL generation for presentation to user, prompts user for information
5. Database Interaction – Receives SQL queries from Dialog Manager, interfaces to SQL database and returns data, Retrieves live data from the web.
6. Text-to-Speech Synthesis – Receives word strings from NL generation, synthesizes them to be sent to the audio server.
7. User Preference – Allows users to enter preferences profile which is used by the Dialog Manager to supply default values for airlines, hotels, cars, etc.

¹ This work supported by DARPA through the Office of Naval Research under grant # N00014-99-1-0418

Servers do not interact directly with each other, but pass frames to the hub. The hub acts as a router which send the frames to the receiving server. A simple script controls hub actions.

2.1 Audio Server

The audio server is responsible for answering the incoming call, playing prompts and recording user input. Initially, we are using the MIT/MITRE Galaxy-II audio server. The telephony hardware consists of a Computerfone¹ with a serial cable connection to the host computer. The record process is pipelined to the speech recognition server and the play process is pipelined the text-to-speech server.

2.2 Speech Recognition

We are currently using the Carnegie Mellon University Sphinx-II system[3] in our speech recognition server. This is a semi-continuous Hidden Markov Model recognizer with a class trigram language model. The recognition server receives the input vectors from the audio server. The recognition server produces a word lattice from which a single best hypothesis is picked and sent to the hub for processing by the dialog manager. We will soon be changing the architecture to allow parser access to the word lattice.

2.3 Natural Language Parser

We use a newly modified version of the Phoenix[5] parser to map the speech recognizer output onto a sequence of semantic frames. A Phoenix frame is a named set of slots, where the slots represent related pieces of information. Each slot has an associated context-free semantic grammar that specifies word string patterns that match the slot. The grammars are compiled into Recursive Transition Networks, which are matched against the recognizer output to fill slots. Each filled slot contains a semantic parse tree with the slot name as root.

Phoenix has been modified to also produce an extracted representation of the parse that maps directly onto the task concept structures. For example, the utterance

“I want to go from Boston to Denver Tuesday morning”

would produce the extracted parse:

```
Flight_Constraint: Depart_Location.City.Boston
Flight_Constraint: Arrive_Location.City.Denver
Flight_Constraints:[Date_Time].[Date].[Day_Name].tuesday
                  [Time_Range].[Period_Of_Day].morning
```

Functions can be associated with roots of sub-trees in order to put the extracted value in canonical form. [Date_Time] is an example of a value that must be transformed. If a token in the parse has an associated transform function, the function is executed and replaces the original sub-tree with a new one. After transformations, assuming the current date is Wed 7/21/1999, the above parse becomes:

```
Flight_Constraint: Depart_Location.City.Boston
Flight_Constraint: Arrive_Location.City.Denver
```

```
Flight_Constraints:[Date].[Month].7 [Day].21
                  [Time_Range].[Start].600
                  [Time_Range].[End].1159
```

2.4 Dialog Manager (DM)

The Dialog Manager controls the system’s interaction with the user and the domain server. It is responsible for deciding what action the system will take at each step in the interaction. The Dialog Manager has several functions. It resolves ambiguities in the current interpretation; Estimates confidence in the extracted information; Clarifies interpretation or context with user if required; Integrates input interpretation with dialog context; Builds database queries (SQL); Sends information to NL generation for presentation to user; and prompts the user for missing information.

A flexible dialog control strategy is essential for robust and graceful interaction with users. We have developed a flexible, event driven dialog manager in which the current context of the system is used to decide what to do next. Systems do not need a dialog script; a general engine operates on the semantic representations and the current context to control the interaction flow. This provides for a robust and portable system.

The Dialog Manager receives the extracted and transformed parse. It then integrates the parse into the current context. The current context is a set of C structures, declared in a header file. This C header file can be automatically generated from a forms file created by the developer. This forms file is the link between parsed structures and the internal context maintained by the system. It specifies a form based concept representation; similar to the concept based translation used by the Janus speech translation system[6] developed at Carnegie Mellon University. Forms are defined by the developer in a “forms” file. Each form has a set of fields and each field has a set of pointers. The first pointer is the field name; the second is to the value extracted for the field. The third is to the parser extracted token string that maps to the field. This provides the link between the parses and context structures. Other pointers are to templates to output the value of the field in various languages, some formal, and some natural. The templates are filled in by the fields from the structure to generate output in the desired language. This is the way we currently generate SQL queries and user prompts in our communicator system. Developing this feature gives us a simple, consistent mechanism for interaction languages, be they languages for interacting with a database (like SQL) or natural languages for interaction with humans. The mechanism is easily extended to be multi-lingual, it could input in any of several languages and output in several, possibly different, languages (any combination of machine and human). An example form specification is:

```
Form: car
Name: company_name
Parse: [Rental_Company]
SQL: "car_comp_name like '%!%'"
English prompt: “What car company would you like”
```

```
Name: car_type
Parse: [Car_Class]
```

¹ <http://sites.gulf.net/synthia/fone.htm>

SQL: "car_type_code in (select rental_code from car_rental_types where rental_type like '%!%')"

English prompt: "What type of car would you like"

Each structure has a set of fields that are pointers to character strings. The dialog manager allocates memory for each leaf string in the parse and puts the pointer in the associated field.

In addition to extracting parsed values to structures, the dialog manager must also set global variables in the context in response to an incoming parse. This is accomplished with the same mechanism as the canonical transforms. C functions are associated with grammar tokens. When a token appears in a parse, its associated function is executed. These are typically very simple functions which just set global variables. An example is [Return], which causes the flight leg number to advance, the current depart_loc to be the arrive_loc from the previous leg, and set the current arrive_loc to the origin. The system developer provides a file with a set of C functions and a table, which associates a grammar token with a pointer to the function.

This "event driven" architecture functions similar to a production system. An incoming parse causes a set of actions to fire which modify the current context. After the parse has been integrated into the current context, the DM examines the context to decide what action to take next. This action is not controlled by a script, but is dynamically driven by the context. The DM attempts the following actions in the priority listed:

- Clarify if necessary – If the interpretation is uncertain or ambiguous, interact with the user to get a unique and high confidence interpretation.
- Sign off if all done – Summarize itinerary and hang up if end of session.
- Retrieve data and present to user – If there is enough information to build a database query, build it and get the results. Send the best row as a suggestion or response to the user. This includes responses to specific queries.
- Prompt user for required information – If there is not enough information to take the desired action (build a DB query) prompt the user for required information.

The rules for deciding what to prompt for next are very straightforward. The system has a variable representing the frame in focus. The frame in focus is set to be the frame produced in response to the user, or to the last system prompt.

- If there are unfilled required slots in the focus frame, then prompt for the highest priority unfilled slot in the frame.
- If there are no unfilled required slots in the focus frame, then prompt for the highest priority missing piece of information in the context.

Our mechanism does not have separate "user initiative" and "system initiative" modes. If the system has enough information to act on, then it does it. If it needs information, then it asks for it. The system never requires that the user respond to the prompt. The user can respond with anything and the system will parse the utterance and set the focus to the resulting frame. This allows the user to drive the dialog, but doesn't require it. The

system prompts are organized locally, at the frame level. The dialog manager or user simply puts a frame in focus, and the system tries to fill it. The mechanism doesn't have a global script and therefore doesn't have to keep track of where it is. This representation is easy to author, there is no separate dialog control specification required. It is also robust in that it has a simple control that has no state to lose track of.

An additional benefit of Dialog Manager mechanism is that it is very largely declarative. Most of the work done by a developer will be the creation of frames, forms and grammars. The great majority of the domain specific information is in external files, not compiled C code. Very little domain specific code has to be written, and they are primarily small, simple functions.

2.5 Database and Web Server

The back-end interface consists of an SQL database and domain-specific Perl scripts for accessing information from the internet. During operation, database requests are transmitted by the Dialog Manager to the database server via a formatted frame. The frame contains key/value pairs, which describe the domain (e.g., airline, car, or hotel information), the access mode, an SQL command, and basic information needed to perform an information retrieval from the internet.

The back-end consists of a static and dynamic information component. Static tables contain data such as conversions between 3-letter airport codes and the city, state, and country of the airport (e.g., BOS for Boston Massachusetts). There are over 8000 airports in our database, 200 hotel chains, and 50 car rental companies. Other static information sources include code values for representing hotel chains, rates, and amenities as well as car rental types (e.g., "luxury" versus "economy" cars). The dynamic information content consists of database tables for car, hotel, and airline flights. Tables of this type are updated from an internet travel information provider during the user interaction.

When a database request is received, the Dialog Manager's SQL command is used to select records in local memory. If no records are found to match, the back-end can optionally submit an HTTP-based request for the information via the internet. Records returned from the internet are then inserted as rows into the local SQL database and the SQL statement is once again applied. A specialized access key, set by the Dialog Manager, is used to block access to the web during certain database accesses.

Our interface to the internet consists of a set of domain specific Perl scripts with HTTP extensions. The Perl scripts perform the functions of logging into the travel web site, submitting the information query, and parsing the resulting HTML pages.

When accessing the internet, we have experience delays ranging from 3 to 6 seconds on average. Depending on the time of day and day of the week, the delays to the internet can be exceptionally long. To cope with this problem, our back-end submits requests in a non-blocking manner and establishes a poll structure, which has a 30-second time-out. If a resulting HTML page is not returned during the time-out period, the user

is notified by the Dialog Manager that excessive delays are being experienced on the internet.

2.6 Text to Speech Synthesis (TTS)

We have developed two Hub-compliant TTS servers for the CSLU Communicator. The first is based on the Festival TTS system[7]. Our TTS server provides an interface between the DARPA Hub and the socket-based connection of the Festival server. Waveforms are synthesized on a per-sentence basis and pipelined to the audio server. Our implementation of the Festival server allows for voice quality, speaking rates, pitch, and synthesis language (English, Spanish or German) to be configurable as part of a Hub script.

In addition to the Festival TTS interface, we have developed a variable unit waveform concatenation synthesizer. Similar in some respects to [8] and [9], our concatenative synthesizer can adjoin units ranging from phonemes, to words, to phrases and sentences.

Data for the concatenative synthesizer were recorded from approximately 850 domain dependent phrases (e.g., "United flight" and "departs Washington at"), 400 phonetically balanced sentences from the TIMIT corpus, and 500 city names. Each utterance is orthographically transcribed and phonetically aligned using a HMM-based recognizer. Our research efforts for data collection are currently focused on methods for reducing the audible distortion at segment boundaries, optimization schemes for prompt generation, as well as tools for rapidly correcting boundary misalignments.

During synthesis, the text is automatically divided into individual sentences. A text-to-phoneme conversion is applied using a phonetic dictionary and letter-to-sound rules. The selection of units to concatenate is determined using a Viterbi search across all available units. The cost function includes information regarding phonetic context, pitch, duration, and signal amplitude. Spectral distance between units is incorporated using a measure based on the Line Spectral Frequency (LSF) parameters[10] (adjacent units in the database are assigned a spectral distance of zero). The forward Viterbi search over the database is carried out using a beam-search. Audio segments making up the best-path are then concatenated to generate the final sentence waveform.

2.7 User Preference Server

A web page interface is used to establish user profiles for the dialog system. The web page allows each user to customize the functionality of the interactive dialog as well as provide a means for sorting airline, hotel, and car rental choices returned from the back-end information source. Specifically, our current implementation allows users to input general information such as the traveler's name, email address, and phone number. For airline flights, users can select seat, meal and airline carrier preference. The user can also configure the dialog manager to return information according to cheapest airfare, exactness of desired departure and arrival times, and airline carrier. Similar functionality exists for hotel and car rental preferences.

The web-page interface also allows users to customize the dialog experience. For example, the user can select between a verbose or terse mode of natural language generation. Finally, each user can customize the text-to-speech synthesis output (e.g., the voice, pitch and speaking rate for the Festival text-to-speech synthesizer).

3. FUTURE WORK

Our dialog model will also provide predictions about user behavior, which will be integrated into the recognition and understanding processes. An initial dialog model for a user will be derived from the general model and the user's preference file. As the user interacts with the system, the system will learn a stochastic model of the user's behavior, which reduces the uncertainty about what the user means or will do next. We propose to use statistical decision trees to adapt the system jointly at multiple levels. These include: acoustic models, language models, pronunciation models, dialog models, and confidence measures.

We expect to field a live system for continuous evaluation before the end of 1999.

4. REFERENCES

- [1] <http://fofoca.mitre.org>
- [2] 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, Vol. 3, pp. 931-934, 1998.
- [3] Ravishankar, M.K., "Efficient Algorithms for Speech Recognition". Unpublished Dissertation CMU-CS-96-138, Carnegie Mellon University, 1996
- [4] Eskenazi, M., Rudnicky, A., Gregory, K., Constantinides, P., Brennan, R., Bennett, K., Allen, J., "Data Collection and Processing in the Carnegie Mellon Communicator," to appear *Proc. Eurospeech-99*, Budapest, Hungary.
- [5] Ward, W., "Extracting Information From Spontaneous Speech", *Proc. ICSLP-94*, September 1994.
- [6] Mayfield, L., Gavalda, M., Ward, W. H., and Waibel, A., "Concept-Based Speech Translation", *Proc. IEEE ICASSP*, May 1995.
- [7] Taylor, P., Black, A., Caley, R., "The Architecture of the Festival Speech Synthesis System," *3rd ESCA Workshop on Speech Synthesis*, Jenolan Caves, Australia, pp. 147-151, 1998.
- [8] Donovan, R.E., Franz, M., Sorensen, J.S., Roukos, S., "Phrase Splicing and Variable Substitution using the IBM Trainable Speech Synthesis System," *Proc. ICASSP-99*, Phoenix AZ, Vol. I, pp. 373-376, 1999.
- [9] Yi, J.R.W., Glass, J.R., "Natural-Sounding Speech Synthesis using Variable-Length Units," *Proc. ICSLP-98*, Sydney Australia, Vol. 4, pp. 1167-1170, 1998.
- [10] Pellom, B.L., Hansen, J.H.L., "Trainable Speech Synthesis based on Trajectory Modeling of Line Spectrum Pair Frequencies," *IEEE Nordic Signal Processing Symposium*, pp. 125-128, Vigso Denmark, June 1998.