THE NATURAL LANGUAGE PROCESSING MODULE FOR A VOICE ASISTED OPERATOR AT TELEFÓNICA I+D

J. Alvarez-Cercadillo, J. Caminero-Gil, C. Crespo-Casas and D. Tapias-Merino

Emilio Vargas 6, 28043 - Madrid, SPAIN.
phone: (34) 1 3374769. fax:(34) 1 3374202. e-mail: jorge@craso.tid.es

ABSTRACT

A Natural Language Processor module has been implemented at Telefónica I+D. This module is composed by a Semantic Parser, a Message Generator, and a Dialog Controller which coordinates all the process.

The Dialog Controller is an inference engine that works through a semantic tree. This tree hierarchically organizes the information going from general concepts, towards more specific ones. The dialog strategy at each tree node can be chosen by selecting the node type from among five pre-defined node types.

Output Messages are selected by the Message Generator. There are six different types of messages and two recovery strategies that the Message Generator can follow. Several output messages for each node in the tree can be defined in order to provide variability.

Based on this scheme, a conversational system call ATOS (Automatic Telephone Operator Service) has been designed at Telefónica I+D. In this document we explain the main characteristics of the system an how every module works.

1. SYSTEM OVERVIEW

Our natural language processor is being used in the first version of the ATOS conversational system. This prototype integrates continuous speech recognition, natural language processing and text to speech conversion. The ATOS is being used for voice commanding some telephone facilities and for information retrieval using telephone and spontaneous speech. ATOS offers four main capabilities: First, by using *the telephone directory* the user can asks for the phone number of all the 833 employees of Telefónica I+D by their names and dial them if desired; they can even specify where to phone them: office, home, cellular phone,... Second, the user have a *personal agenda* available, where he can keep the most frequent phone numbers; ATOS offers capabilities to voice command the directory functions as include a new phone and person, or delete an old one. Third, the user can ask for deleting, recording or listening to the voice messages in his *answering machine.* Finally ATOS accepts voice commands and its parameters for executing all the *services provided by a PABX*, as call somebody, switch the current call to another number, recall last number, do not disturb, follow-on call, etc. A block diagram of ATOS is presented in Figure 1. In this paper we explain how the system works.

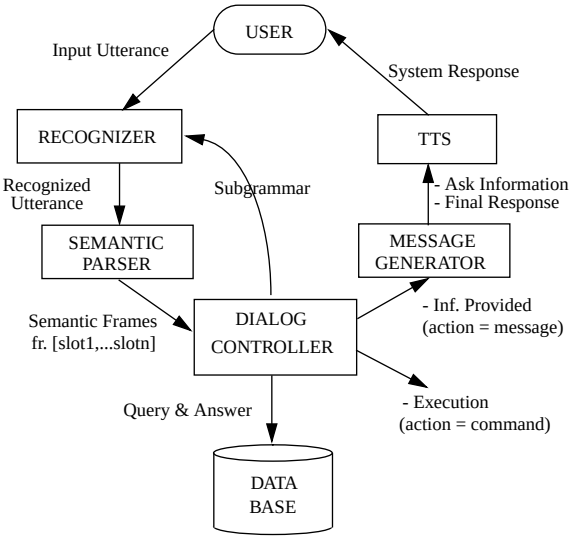

Figure 1: Block Diagram of ATOS

The recognition module. The recognizer takes the speech from the user and gives as result a chain of words. This module is a speaker independent continuous speech recognizer based on the CMU Sphinx system. The word accuracy rate of the continuous speech recognizer is 95% in laboratory conditions. The main errors of the system are due to gender and number disagreement, and insertions or deletions of function words. Most of this errors do not change the meaning of the sentence, so that the semantic parser and the dialog controller are able to overcome them successfully.

The semantic parser. The parser is a modified version of the Phoenix system of C.M.U. [Issar, 93]. This is a flexible frame-based parser, which is based on semantic frames which represent forms. Semantic frames are composed by semantic slots. These semantic slots are hierarchically structured and represent the basic semantic entities known the system. The parser operates by matching these slots with sub-strings in the sentence. These sub-strings will be named as slots if they match the Recursive Transition Networks (RTN) for that semantic slot [Allen, 87].

In our system we have modified the semantic parser in order to allow word insertions at any part of the RTN. This ability is very useful to make the system robust from word insertions, either from

the recognizer or from the speaker who usually includes non-keyword expressions in spontaneous speech. Moreover, this parse strategy considerably simplifies the definition of the RTN´s, because it is not necessary to explicitly define all the possible combinations of words that appear between two keywords.

Before parsing, a concept clustering is done in order to simplify the parse and reduce the parser process time. In Spanish infinitives are multiply inflected in tense, number, person and gender. This means that we can find lots of surface representations for a same deep semantic representation. As example we can say that every infinitive can appear in 50 different inflections in mean. Specifying all inflections for every infinitive is a tedious task and introduces an important delay in the parsing process. Usually all inflections of an infinitive comprise the same concept, i.e. the number or person of the verb rarely is important when we already know the agent of the action. For instance, when we know that the agent is the second singular person "you", the same understanding can be inferred from both sentences "Send me a bill" or "Sends me a bill" even if there were a recognition error in the word "sends" in the second sentence. So, in some cases a previous clustering of verb inflections can be done before semantic parsing the sentence. Similar clustering can be done with nouns and adjectives depending on their gender and/ or number. Based on this property, we include a clustering module before the semantic parser that both, significantly reduces the parser processing time, and notably simplifies the slot definition stage.

The Dialog Controller (DC) is the engine of the system. It takes all the parsed semantic slots as input, and tries to fill the slots of the frame that are related with the current dialog state. If any form is completed and that form is a command, then the DC orders to execute the command; but if the form is an information request, the DC orders to extract the information, and pass the result to the **Message Generator** (MG). If the form is not completed, the MG selects the most proper message between all messages proposed by the DC. Once the best message is selected at each moment, the message is passed to a Text To Speech (TTS) Converter [Rodríguez, 93] which utters the message. A more detailed description of the Dialog Controller and Message Generator will be given in next section.

2. DIALOG CONTROLLER

One of the most important objectives of our DC [5] has been the implementation of a collaborative dialog model. So the system has to be able to understand all the user actions in whatever order they come, and even if the focus of the dialog has been changed by the user. To achieve this, it is necessary to organize the information in a hierarchical tree, and let the data drive the dialog.

2.1. Hierarchical trees

If we want to build a question-answering system, we have to use two sources of information: The meaning of the sentence said by the user, and the world knowledge embedded in a database and/or in the structure of the task. To extract the meaning of the sentence, the parser splits the sentence in slots, but is the DC who is in charge of giving a meaning to the slots, and in charge of continuing the dialog; the meaning for each slot and how to continue the dialog are given to the DC by the semantic hierarchies that describe our world, i.e. the information embedded in the database. For instance see in Figure 2 shows the semantic tree we have constructed to represent the information necessary to extract a new message from a particular voice mail-box. Nodes of the tree are semantic concepts that tight other related semantic concepts, so the root node is the general semantic concept that tights all the forms in the application. Links of the tree represent semantic relationships between concepts. Finally leaves of the tree represent semantic concepts too, and they are filled if some slot attached to this leave is matched.

In Figure 2, the root node represents the global task. The first level node "Read_message" is one of the forms to be filled. The "read" function plays a message from a particular mail box. The nodes "mail_id" and "mailbox_id" represent the slots of the form; in this case one slot refers to the identification of the mail box user wants to listen to. The other slot refers to the identity of the message in the mail box. The branches "new_mail" and "recorded-mail" coming from the "mail_id" node represent two possibilities for identifying the message to be listened: listen to "new messages", or listen to previously "recorded-messages". The branches "hour" and "order" coming from both nodes allow users to select a message in several ways; i.e. based on the recording hour period, the arriving order, etc.

In this structure all the information necessary to ask for a message to be read is organized. If we want to expand the system with new functions as "delete a message" or "recall", we only have to define similar trees as pointed to in Figure 2. All the information needed to make the system work is specified in a Configuration File. This file defines several features for each node: *type of node,* from among all node types defined in Figure 3, which defines the dialog to be followed at each node (if it will ask for confirmation, for a node value, for the value of its sons, etc...); *relationships* between other nodes defining interactions, dependencies,...; *rules* to be applied at that node, and *messages* to be taken out at each point of the dialog.

2.2. Message Generation

As pointed to in previous section, all the messages that are necessary for the resolution of all the nodes have been specified in the configuration file. Types, tenses and output strategies, are explained in previous work [5].

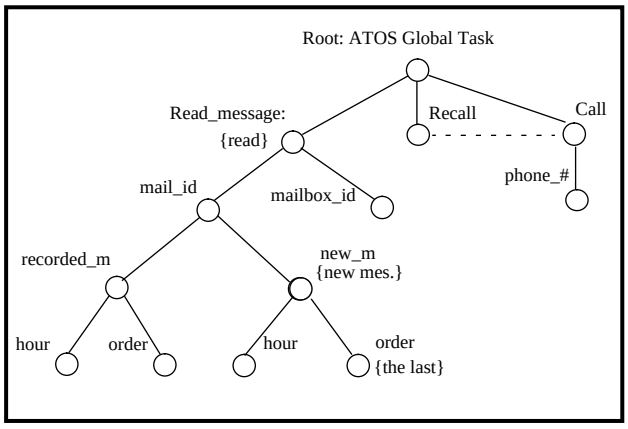

Figure 2: Semantic Tree Example

After the DC has processed all the slots in the sentence, the MG is called. The MG has to select just one message from among all messages in the queue. First, all *conclusion* messages are extracted because they just present extra information and do not ask for new information. Then, if there are some messages in the queue asking for *confirmation*, the most recent one is taken out. Sometimes the queue is empty when the MG is called. For those cases the MG has to find an acceptable question too. So the MG goes to the active node in the tree and takes its proper message out depending on which state the active node is at that moment. Once the MG finds out the adequate type of message for a node (conclusion, asking confirmation or asking a value), it selects the proper tense (past, recent and distant) for the message, and the proper kind of message (alternate, hierarchical) depending on when the node was last acceded and the kind of node it is. Following this simple strategy we are providing good answers at each dialog cycle.

2.3. Data-Driven Dialog Controller

The key point to manage the dialog controller to be task independent is to organize the information in hierarchical trees. However, when implementing a real task, a monotonous behavior of the DC at each node is not desirable. Otherwise we would like to be able to define different types of nodes depending on how we would like the tree to be filled, and therefore how we would like the DC to behave. For this reason different types of nodes can be defined in the configuration file: *necessary nodes* that have to be filled, *optional nodes* that can be not filled because they just give extra information, two or more *complementary nodes* that require just one of them to be filled, and nodes that *request for confirmation*. Moreover we should distinguish among nodes without sons, that ask for its own value, from nodes with sons that ask for the values of their sons. Depending on the type of node, the DC will behave different according to the state diagram in Figure 3. For example the C node is defined to ask for confirmation of its value. When the DC receives a slot that matches the C node, and the C node is the "EMPTY" state, then the DC fills in the node with the slot, and puts the node in the "WAITING FOR CONFIRMATION" state. Following, the DC will ask the user to confirm the node value. This is done during the transition from the EMPTY to the "WAITING FOR CONFIRMATION" state; in this transition, a "message for confirmation" for this node is put in a queue of output messages. This queue will have to be processed by the Message Generator later. However if the received slot does not match the C node, or the node is not in the EMPTY state, the slot is rejected for this leave of the tree. Therefore this strategy makes the system to be more robust from recognition errors.

Moreover we can define some rules for each node to modify the default behavior of the DC. The two main rules allowed are *Implication* and *Restriction*. If a node has an *implication rule*, then when this node is filled, the value of another node is automatically filled too. If a node has a *restriction rule*, the value is checked before being validated. Some considered restrictions are quantitative comparative (mayor, equal,...), and list ordering (after, before,...).

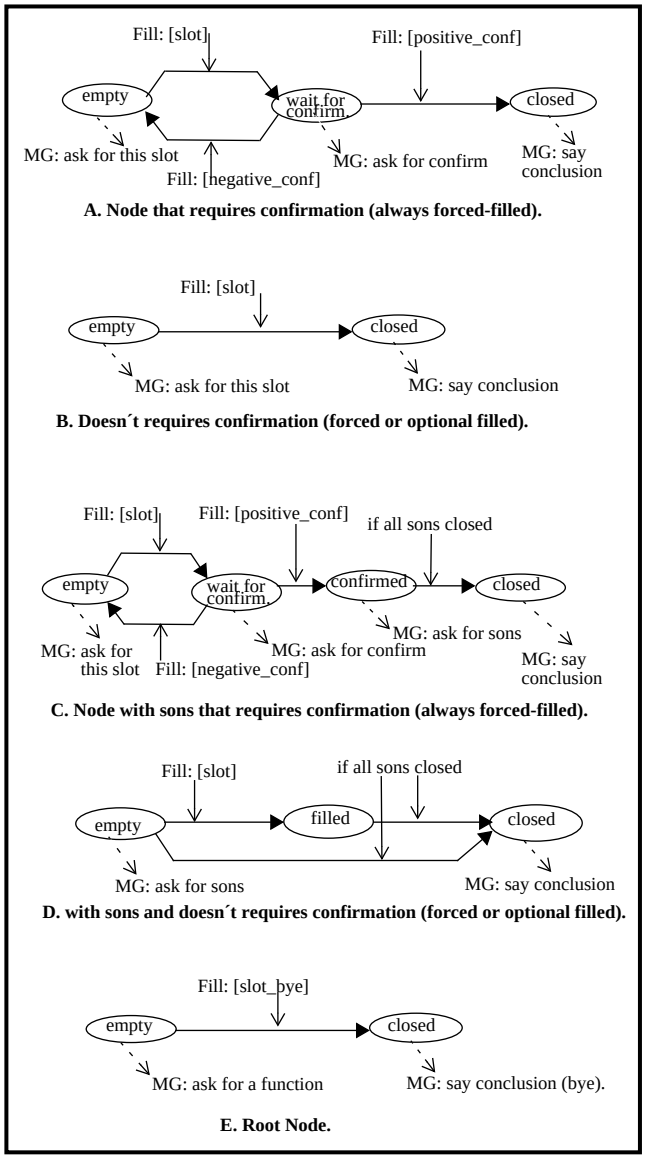

Figure 3: State Diagram for basic types of nodes

Using all these information sources, the behavior of the DC is as follows. When the system is initialized, the root node in Figure 2 is the active one, and a message from this node is taken out. When a sentence is uttered, the parser splits it in one or several slots. The DC tries to fill every node in the tree using every parsed slot, starting from the active node and continuing with all the nodes in the scope. When a slot matches a node, the DC fills in that node, puts it as the active node, changes the state of the node according to the states in Figure 3, and puts a potential message in a lifo queue that will be processed by the MG. After the DC have processed all the slots in the sentence, the MG is called. This MG is in charge of taken out just one message at each dialog cycle, selecting just one from all the messages in the queue.

Confirmations and Negations. As can be seen in Figure 3, the dialog in nodes A and C goes through a special state called "wait_for_confirm". Being in a node with this state implies that the DC is waiting for either a [confirm] or [reject] slot, so the DC is in a special state. These slots are free to appear anywhere in the user response. If the user answers just using "yes" or "no", parsing and processing is very simple. But usually some extra information for clarification is given after negation; for instance in the sentence "don't delete the message, but read it", two coordinate functions underlay: first, the rejection of the "delete" function, and second, the acceptance of the "read" function. Similar double information is able to appear in confirmation sentences too; for instance in the sentence "yes, and delete it", we should confirm the previous function, and accept other function too (i.e. read the message, and delete it later). For dealing with this capability when the dialog is in a "WAIT FOR CONFIRM" state, the DC scans all the parsed sentence in search of either a [confirm] or a [reject] slot. If found, the DC processes these special slots first and confirms or rejects the present value; then the DC continues processing the rest of the sentence. Based on this strategy, we are able to process several coordinate actions in an utterance; of course, all coordinate actions appearing in an utterance are processed in the same dialog cycle too, even if the utterance has not any confirmation or rejection slot.

Anaphora and Ellipses. Some linguistic phenomena as ellipses and anaphora have to be resolved during human-machine dialog. In order to interpret them we have implemented a simple structure called *historic*, that maintains the necessary context information. Figure 4 shows an example of how we represent the historic. Associated to every human utterance a new element is linked to the historic list. Each element includes:

- The complete recognized utterance (RU).
- The parser interpretation (PI), composed by all parsed slots and the *focus of the dialog*.
- Interpretation (I) after anaphora an ellipses resolution.
- Interpretation (GI) at the moment; based in previous sentences related with the present dialog subject.
- The system response (R) to the human.

Based on this representation, anaphora resolution procedure is simple. If an anaphoric slot is detected when attempting to fill all slots in a function, then the historic list is scanned back in search of that slot which completes the present function in accordance with the rest of filled slots. For instance, in the second element of Figure 4, we are attempting to resolve the function [erase] that is pre-defined as having a parameter [message_id] which identifies the needed message. When an anaphoric slot [anaphora] is found (it) while completing the [erase] function, the system has to resolve this anaphora by searching the historic list for a [message_id] slot that completes the present function [erase]. As a [message_id] slot with the value (last message) was filled in the previous element of the historic, this [message_id] slot is taken and the present function [erase] is competed. Some ellipses situations are solved following a similar procedure.

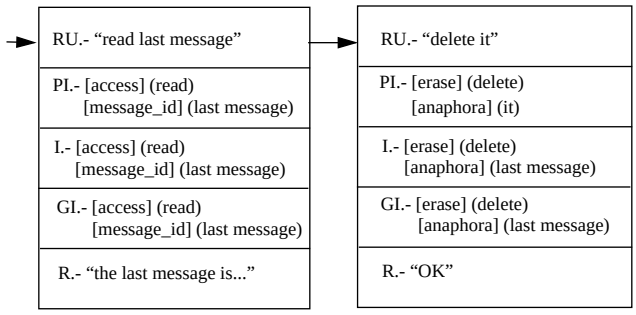

Figure 4: An example about the dialog historic representation.

3. EVALUATION

The Voice Assisted Operator has been tested using the Telefónica I+D Continuous Speech Recognizer, and its TTS system. Within the laboratory we have demonstrated flexible and robust dialogs.

To evaluate the system a total of 20 male and 20 female subjects were asked to attempt five of our 26 possible functions. In total each function was attempted more than four times. Subjects were given no prior training or information on how to use the system.

The average completion rate for all the tasks was 84% for males and 72% for females. These results are strongly dependent on two factors: recognition and parser errors. The overall word error rate of the recognizer was 19.3% with a corresponding 72.1% sentence error rate. This is one of the most important sources of errors. Besides this, about 51% of the sentences had not a full parse. If we analyze this set of sentences we can see that even not having a full parse, the 76% of the semantic concepts were correct, and another 24% were partially correct, but most of the times the system was able to recover from these wrong parsed sentences.

ACKNOWLEDGEMENTS

The authors are grateful to Ravishankar Mosur (CMU) and Eric Thayer (CMU) for their support and help in using the SPHINX system. We also wish to thank Wayne Ward (CMU) and Sunil Issar (CMU) for their support in using the PHOENIX system.

4. REFERENCES

1. Issar, S. and Ward, W. "CMU's Robust Spoken Understanding System". *Proc. Eurospeech'93*. Berlin.
2. Young, S.J. and Proctor C.E. "The Design and Implementation of Dialogue Control in Voice Operated Database Inquiry Systems". *Computer Speech and Language (1989) 3, Pag. 329-353.*
3. Rodríguez, M.A., Escalada, J.G., Macarrón, A. and Monzón, L. "AMIGO: Un Conversor Texto-Voz para el Español". *SEPLN VIII Congress.* Barcelona. Feb 93.
4. Caminero-Gil, J., Alvarez-Cercadillo, J., Crespo-Casas, C., and Tapias-Merino, D. "Data-Driven Discourse Modelling for Semantic Interpretation". *ICASSP-96*. Atlanta. USA.