Real-time linguistic analysis for continuous speech understanding*

Paolo Baggia Elisabetta Gerbino Egidio Giachin Claudio Rullent

CSELT - Centro Studi e Laboratori Telecomunicazioni
Via Reiss Romoli, 274 - 10148 Torino, Italy

Abstract

This paper describes the approach followed in the development of the linguistic processor of the continuous speech dialog system implemented at our labs. The application scenario (voice-based information retrieval service over the telephone) poses severe specifications to the system: it has to be speaker-independent, to deal with noisy and corrupted speech, and to work in real time. To cope with these types of applications requires to improve both efficiency and accuracy. At present, the system accepts telephone-quality speech (utterances referring to an electronic mailbox access, recorded through a PABX) and, in the speaker-independent configuration, it correctly understands 72% of the utterances in about twice real time. Experimental results are discussed, as obtained from an implementation of the system on a Sun SparcStation 1 using the C language.

1 Introduction

We call *continuous* speech, as opposed to *isolated-word* speech, any utterance emitted without interposing pauses between words. This is the way humans naturally speak, but to enable a machine to deal with this form of communication constitutes a difficult task because, in addition to the usual speech processing and natural language issues, there is no hint on where the single words of the utterance begin and end. Given the current state-of-the-art, speech understanding prototypes concern the comprehension of utterances referring to a well defined semantic domain on a dictionary of almost one thousand words.

Our research group is interested in developing automated voice-based information retrieval services over the telephone network. This has some precise implications. First, we are committed to accept continuous-speech sentences expressed in relatively free syntax, otherwise the interaction with the service would be too unnatural. Second, the service should be public and accessible from every telephone; this means that the understanding system has to be *speaker independent* and able to process noisy and distorted speech. Third, the response time must be confined within a few seconds; that is, the system has to work in *real time*.

The evolution of the research is gradual. In the present status of work the developed system is speaker independent and it is based on a telephone-line quality speech (a telephone connected through a PABX). It has a vocabulary of 787 words and the processing time is less than 5 seconds (about twice real time). The sentence correct understanding rate is nearly 72%. The application task refers to voice access to an electronic mailbox. In its first version the system has been applied to access a geographical data base, and is now adapted (with a slightly bigger vocabulary) to information retrieval from a train timetable data base. In the following we present a thorough description of the approach that led to these results. Also, the latest developments of the system are discussed. We will focus here on the understanding subsystem. An account of the recognition stage is given in [Fissore *et al.* 1989] and in the references there reported, while the whole system is described in [Baggia *et al.* 1991c]. We will examine the role of the recognition and understanding modules, the technique used for language representation, the parsing control strategy, and finally experimental results will be discussed.

2 Recognition and understanding activities

Speech understanding requires the use of different pieces of knowledge. Consequently, it is not obvious a priori what type of architecture will give the best results. Homogeneous, knowledge-based architectures date back to the late 1970s [Erman *et al.* 1980] and spurred interesting research work in the subsequent years. However, unified approaches contain a weakness: they have difficulty in coping with problems of different nature through specific, focused techniques. A division may be traced between lower-level processing of speech, mostly based on acoustical knowledge, and upper-level processing, mostly based on natural language knowledge. Therefore, a two-level architecture has been developed based on this idea [Fissore *et al.* 1988]. The former stage, called *recognition stage* (Fig. 1a), hypothesizes a set of words all over the utterance and feeds the lat-

*This research has been partially supported by EEC ESPRIT project no. 2218 SUNDIAL.

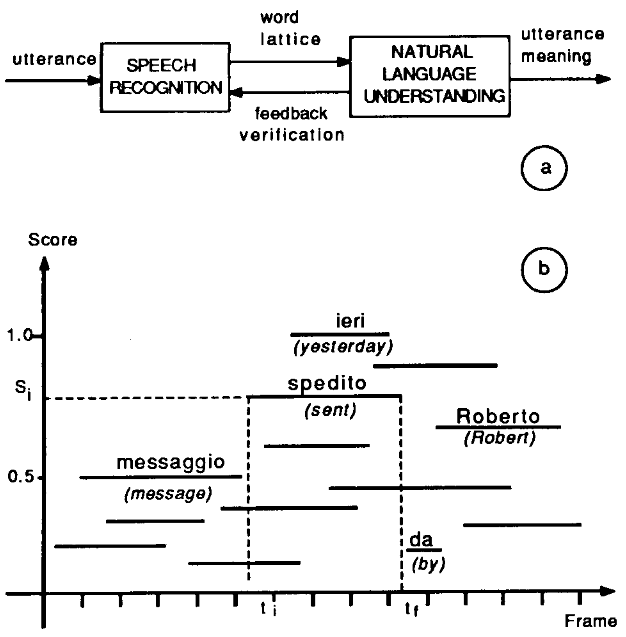

Figure 1: System architecture (a). An example of word lattice (b).

ter stage, or ***understanding stage***, which completes the recognition activity by finding the most plausible word sequence and by understanding its meaning. In this way each level can focus on its own basic problems and develop specific techniques, still maintaining the advantage of the integration.

Most of the approaches based on this idea (e.g. [Hayes *et al.* 1986]) are characterized by the use of knowledge engineering techniques at both levels, while our recognition stage is based on a probabilistic technique, the hidden Markov models (HMM). The most recent research indicates that, as far as word recognition is concerned, the HMM give the best results [Lee 1990, Fissore *et al.* 1989].

The set of word hypotheses produced by the recognition stage is called *lattice* (Fig. 1b). Every word hypothesis is characterized by the starting and ending points of the utterance portion in which it has been spotted, and its *score*, expressing its acoustic likelihood, i.e. a measure of the probability for the word of having been uttered in that position. Many more hypotheses than the actually uttered words are present in the lattice (there are about 30 times as many word hypotheses as there are words), and they are overlapping on one another. The aim of the understanding stage is then twofold: on one side it has to complete the recognition task by extracting the correct word sequence out of the lattice; on the other it has to understand the sequence meaning. In practice these two activities are performed simultaneously. The correct word sequence extracted by the understanding stage may be fed back to the recognizer (Fig. 1a) for a post-processing phase called ***feedback verification***, described below, aimed at increasing the understanding accuracy.

The problem of analyzing lattices is considered from the natural language perspective: the goal is to develop techniques to process typed input and to extend them in order to process a "corrupted" form of input such as a lattice is. The understanding stage result called ***solution*** is a sequence of word hypotheses spanning the whole utterance time so that 1) the sentence is syntactically correct and meaningful according to the linguistic knowledge of the understanding stage, and 2) it has the best acoustical score among all of the possible sequences that satisfy point 1). The great problem is that the search for a solution cannot be made exhaustively: since the lattice contains many incorrect word hypotheses, there would be far too many admissible word combinations to examine. In addition there is the risk of incorrect understanding due to the possible selection of even only one incorrect word hypothesis. Coping with them imposes to carefully design linguistic knowledge representation methods and analysis control strategies in order to gain in both efficiency and correct understanding reliability.

3 Language representation

The task of the machine is to combine together adjacent word hypotheses, so as to create ***phrase hypotheses*** (PHs), which are consistent according to the language model. Such parsing process continues until the system reaches a ***solution***.

The choice of a suitable linguistic knowledge representation poses a dilemma. For the machine, to reach real-time, the representation must above all be efficient that is, it must require reasonable computational cost and must keep low the number of PHs generated during parsing. On the other hand, for the developer of the system the representation must be easy to declare interpret, and maintain. Ease of maintenance suggests for example, that it is preferable to keep syntax and semantics separate as much as possible.

The previous considerations suggest to adopt two representations, one suitable for the system developer, the other for the machine [Poesio and Rullent 1987]. The translation of the linguistic knowledge from the former representation (high-level representation) to the latter one (low-level representation) is performed off-line by a ***compiler*** (see Fig. 2). This approach also permits to maintain separate high-level representations for syntax and for semantics, choosing for each the formalism that seem most suitable. For semantics the Caseframe formalism [Fillmore 1968] in the form of Conceptual Graphs [Sowa 1984] had been chosen, while for syntax the Dependency Grammar formalism [Hays 1964] has been used. A Dependency Grammar expresses the syntactic structure of sentences through rules involving dependencies between morphological categories. The right-hand side of the rule contains one distinguished terminal symbol called *governor*, while the other symbols are called ***dependents***. A mechanism has been added to the dependency rules to describe the morphological agreements between the governor and the dependents.

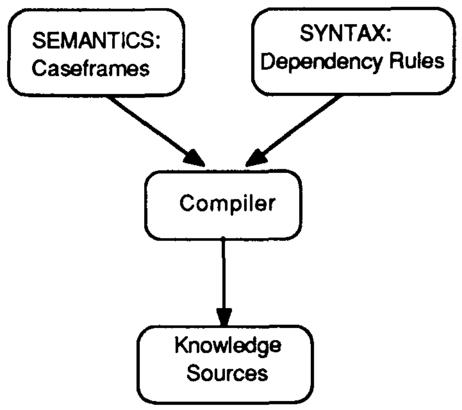

Figure 2: Language representation

3.1 The compiler

Dependency grammars have been selected as a formalism for representing syntactic knowledge because they allow an easy integration with caseframes thanks to the similar notion of governor for the dependency rules and of header for the caseframes.

The compiler operates off-line and generates internal structures, called Knowledge Sources (KSs) suitable to allow an efficient parsing strategy. The basic point is that each KS is aimed at generating a certain class of constituents. Then each KS must combine the time adjacency knowledge, the syntactic, morphological and semantic knowledge that it is necessary to handle a specific class of phrases.

As an example, Table 1b represents the dependency rules used to deal with the sentences of Table 1a. Note that prepositions are never governors as they are usually short and are likely to be missing from the lattice (see section 5). The star symbol in each rule represents the governor position. The associated rules for morphological agreement checks are not reported for simplicity.

(a)		sent yesterday sent yesterday by John
(b)	rs1)	verb = * adverb[adv-phrase]
	rs2)	verb = * adverb[adv-phrase] noun[by-phrase]
	rs3)	noun = prep *

Table 1: An example of phrases (a). The necessary dependency rules (b)

For each dependency rule the compiler must find all the conceptual graphs that can be associated to such rule and to use them to generate a KS. For this purpose, each dependency rule is augmented with information about grammatical relations, contained in square brakets in Table 1b; a grammatical relation is associated

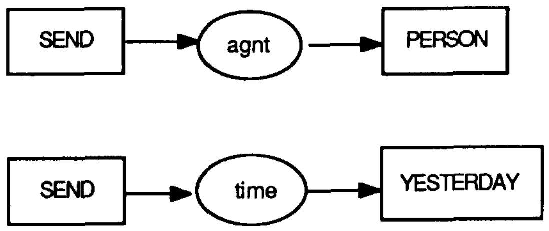

Figure 3: Conceptual graphs

to each dependent D_i, accounting for the grammatical relation existing between the governor G and the lower-level constituent having D_i as a governor.

For example, the associated grammatical relations for rs2 could be ***adv-phrase*** for the first dependent and ***by-phrase*** for the second one. Additional mapping knowledge associates one or more conceptual graphs to each grammatical relation, so that it is possible to find from the conceptual graphs the semantic constraints that the governor and the dependents of the rule have to follow. Referring to the conceptual graphs of Fig. 3, the conceptual relation ***agnt*** can be associated to ***by-phrase*** and the conceptual relation ***time*** can be associated to ***adv-phrase***. The semantic constraints derived from the conceptual graphs are: SEND for the "verb" governor, YESTERDAY for the "adverb" dependent and PERSON for the "noun" dependent of rule rs2.

Each KS built by the compiler has one terminal slot called header, representing one single word, and other slots called fillers representing phrases, positionally reflecting the symbols of the dependency rules from which the KS derives. The main bulk of knowledge embedded in a KS is a set of structures that express constraints between the syntactic-semantic features of the header and those of the fillers.

A first version of the compiler had the goal of enriching the dependency rules with the semantic constraints derived from the concepual graphs. In this case the set of generated KS could be sketched as in Table 3, where row c4 can correspond, for instance, to the compilation of the dependency rule rs2. A total of 70 conceptual graphs and 373 syntactic rules is used in the system. These knowledge bases are able to treat a large variety of sentences, a sample of which is shown in Table 2 (nearly literal English translation).

Although the obtained efficiency was sufficiently good, a conceptual improvement to the compiler has been devised, as is described in the next subsection.

3.2 Efficient representation of linguistic constraints: rule fusion

One basic problem to move towards real-time operation is to define the kind of structures that can be build for representing PHs. Suppose a "classical" grammar like a context free grammar is used and that we are trying to connect two words into a grammatical structure. In general, this can be done in several ways according to

1	I'd like to know Thursday's fourth one.
2	Did any mail come in September?
3	Tell me the mails I received since two days ago.
4	Got any mail last week?
5	Read me Giorgio's first two messages.
6	The third one.
7	Did anyone write after last Friday?
8	Make me a list of your mails.
9	Send SIP's first one to Cselt.
10	What messages did Luciano write me from December one to six?
11	Tell me the senders of messages received from Milan.
12	What are the mails received from Piero of Cselt after October seventeen.

Table 2: A sample of task sentences

different grammar rules. Since structures built with different rules may connect with different word hypotheses, a new memory object is needed for every structure. In the case of speech this leads to two undesirable consequences. First, a very large memory size is required, owing to the high number of word combinations allowed by word lattices. Second, each of the structures will be separately selected and expanded, possibly with the same words, during the score-guided analysis, thus introducing redundant work. Therefore, the compiler should generate a smaller number of "compact" KSs, still keeping the maximum discrimination power.

The goal of generating a small number of KSs is accomplished through the *fusion technique* [Baggia *et al.* 1991a]. Fusion aims at compacting together KSs. KSs

row	position		
	0	1	2
c1	C	A1	
c2	C	B1	
c3	C	A1	B1
c4	C	B1	A1
c5	C	A2	
c6	C	A2	B1
c7	C	B1	A2

Table 3: A sketchy representation of KSs

may have constituents in different order or even a different number of constituents. Let us suppose we have a WH of class C and we want to connect it to other words that can depend on it and that are adjacent to the header on the right. Table 3 contains, for the header class C, a sketchy representation of the KSs involved (the rows in the table). The positions of the constituents are also shown. The zero position indicates the header while positions 1 and 2 indicate dependents on the right of the header. The numbers attached to each class mean that different constraints act on the corresponding constituent. Table 3 shows that constituents of both classes A and B are involved. Let us focus on the class A case.

As we want to find class A constituents, on the right of the header, four KSs are involved, corresponding to rows c1, c3, c5, and c6; the first two KSs propagate constraints (summarized by A1) that will be considered by a proper KS of class A; the result is the generation of two couples of PHs (two generated by the A KS and two by the C KS). Two other couples of PHs are generated in a completely similar way by the KSs of row c5 and c6, the only difference being that the KSs propagate different constraints.

In the fusion case there is just one KS for the seven different rows of Table 3. The C KS propagates the constraints for the A KS: it propagates A1+A2 and the time constraint that the constituent must be adjacent (on the right) to the header. Only one search into the lattice is performed by the A KS. Only a couple of PH is created for the rows c1, c3, c5, and c6 (one by the A KS and one by the C KS).

The fusion technique is effective in reducing the number of PHs to be generated and the parsing time. The results of the experiments are reported in Table 4.

	No Fusion	Fusion
No.PHs generated	806	91
Parsing time (s)	1.56	0.38

Table 4: The effect of fusion

The reduction of PHs would be of no use if it were balanced by an increased activity for checking and propagating constraints. So, for execution efficiency, bit coded representations are used for the propagation of constraints about active rules, in a way similar to the propagation of morphological and semantic constraints. The system runs on a Sun SparcStation 1 and is implemented using the C language, which furtherly increases speed.

4 Control of parsing activities

The basic problem that control is called to face is the width of the search space, due to the combined effect of the non-determinism of the language model and the uncertainty and redundancy of the input. Since an exhaustive search is not feasible, scores are used to constrain it along the most promising directions just from the beginning: the analysis proceeds in cycles in a best-first perspective and at each cycle the parser processes the best-scored element produced so far. The score of a PH made up by a number of word hypotheses is defined as the average of the scores of its component words, weighted by their time durations. This "length normalization" insures that, when we have to compare two PHs having different length, we do not privilege longer or shorter ones.

The building of parse trees may proceed through *top-down* or *bottom-up* steps. For instance, if the best-scored element selected in one cycle is a header word, a top-down step consists in hypothesizing fillers and verifying the presence in the lattice of words that can support them. Hypothesizing headers from already parsed fillers is an example of a bottom-up step.

If all of the correct word hypotheses are well-scored, any parsing strategy works satisfactorily. However, often a correct word happens to be badly recognized and hence receives a bad score, though the overall sentence score remains good. This can be due to a burst of noise, or to the fact that the word was badly uttered. Many incorrect words will be present in the lattice, scoring better than such word. Now, imagine a pure top-down parsing in the case where such a bad word is one of the headers. Prior to processing that header, the parser will process all of the better-scored words that are themselves headers. This may delay the finding of the correct solution beyond reasonable limits, or may favor the finding of a wrong solution in the meantime. Similar considerations hold in the case of a pure bottom-up strategy.

Such bottlenecks are avoided thanks to a strategy in which the good-scored words of the correct solution may hypothesize the few bad-scored ones in any case. This property implies that the parser must be able to dynamically switch from top-down steps to bottom-up steps and vice versa, according to the characteristics of the element that has been selected in that cycle. Apart from avoiding bottlenecks, a control strategy that follows this guideline has one important characteristic: it is *admissible*, that is the first-found solution is surely the best-scored one.

This approach of exploiting only *language constraints*, if followed to its extremes, leads to an insufficient exploitation of *time adjacency*, which is a different criterion for designing an efficient control strategy. Time adjacency is at the base of the so-called island-driven parsing approaches, which recently received renewed attention [Stock *et al.* 1989]. Here the idea is to select only fillers that are temporally adjacent to the header, so that we can limit the number of word hypotheses that can be extracted from the lattice (i.e. that satisfy language and time adjacency constraints) and consequently the parse trees that have to be generated.

The parsing process proceeds through elementary activities, or operators, that represent top-down steps (EXPAND and FILL operators) or bottom-up steps (ACTIVATE and PREDICT operators). The JOIN operator describes the activity in which a KS merges together parsing processes that had evolved separately; this may correspond either to a bottom-up or to a top-down step.

By suitably defining when and how the KSs apply the operators it is possible to trade off with the *language constraint* and the *time adjacency* criteria with the result of switching down admissibility by a little amount while simultaneously gain a consistent reduction of the number of generated parse trees. The control strategy that has been adopted, described in detail in [Giachin and Rullent 1990], accepts a limited risk of getting the wrong solution in the first place (about 1.5%) but is balanced by a great speed-up in the parsing of a lattice.

5 Coping with special speech problems

The adjacency between consecutive word hypotheses is seldom perfect, being them affected by a certain amount of gap or overlap. This is due to the fact that the end of a word is slightly confused with the beginning of the consecutive word. The understanding level is tolerant towards these phenomena and defines thresholds on maximum allowed gap or overlap between supposedly consecutive words.

While coarticulation affects all words, it severely compromises the recognition of what are currently called, with an admittedly imprecise term, *function words*. Function words, such as articles, prepositions, etc., are generally short and they tend to be uttered very imprecisely, so that often they are not included in the lattice. Moreover, function words are often acoustically included in longer words. The parsing strategy then does not rely on function words [Giachin and Rullent 1988]. The idea is that KS slots corresponding to function words are divided into three categories, namely *short*, *long*, and *unknown*. Short words are never searched in the lattice, and a placeholder is put in the Phrase Hypothesis (PH) that includes it. Long words are always searched, and failure is declared if no one is found. Unknown words are searched, but a placeholder may be put in the PH if some conditions are met. In a first phase, the categorization of a KS slot was made on the basis of the morphological features of the corresponding function words and on their length (e.g., words with one or two phonemes were declared "short" and never searched). Subsequent experiments showed that, unexpectedly, some very short words may be recognized with virtually no errors, while others, though longer, are much more difficult to recognize. Hence, better results have been obtained when the categorization has been made on the basis of the phonetic features of the words rather than of the morphological ones.

5.1 Feedback verification procedure

Though skipping function words permits to successfully analyze sentences for which these words were not detected, it also implies that the acoustic information of small portions of the waveform is not exploited, and this may lead the parser to find a wrong solution. Also, function words may be sometimes essential to correctly understand the meaning of a sentence. In order to cope with these problems, a two-way interaction between the recognition module and the parser has been investigated, called *feedback verification procedure* [Baggia *et al.* 1991b]. According to this procedure, the parser, instead of stopping at the first solution, continues to run until a predefined amount of resources is consumed. During this period many different solutions are found, possibly containing multiple possibilities in place of missing words. These solutions are then fed back to the recognizer which analyzes them sequentially. The recognizer task realigns the solutions against the acoustic data and attributes them a new likelihood score. The best-scored solution is then selected as the correct one.

As a side effect, the best-matching candidate for function words that were missing in the lattice is also found.

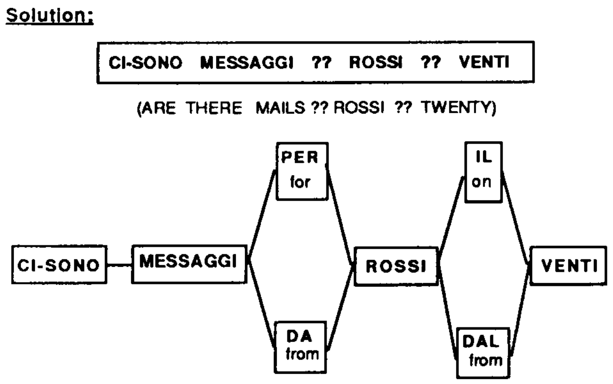
Solution:
CI-SONO MESSAGGI ?? ROSSI ?? VENTI
(ARE THERE MAILS ?? ROSSI ?? TWENTY)
PER for
IL on
CI-SONO
MESSAGGI
ROSSI
VENTI
DA from
DAL from

Figure 4: Function word detection during FVP

The verification procedure creates the best conditions to find these words with good reliability: for each placeholder a very small number of candidates are proposed, and the previous and following words are usually normal reliable words. Hence the recognizer can detect the word with good accuracy. An example of a solution generated by the parser for the utterance "Ci sono messaggi da Rossi il venti?" (literally: "There are mails from Rossi on twenty?") is shown in Fig. 4. The "??" symbol in the solution represents a possibly missing function word ignored during parsing, that is expanded into a set of candidates, according to the grammar, to be fed back to the recognizer.

In addition to accurately finding function words, the verification procedure has the advantage that the final scores assigned to solutions by the recognizer are more accurate than those assigned to them by the parser, because these scores have been computed on the same time interval after a global realignment of the sentences. Hence comparing the solutions on the basis of their score is a more reliable procedure. The drawback of the verification procedure is that total analysis times are slightly increased by the overload imposed to the recognizer and by the fact that the parser must continue the analysis after the first solution is found.

6 Experimental results

In order to evaluate the performance of a speech understanding system it is necessary to define some metric. Unfortunately, metrics are still far from standards in this field. Let us briefly describe the measures used in our evaluation and shown in Table 5. *Understood* refers to the percentage of correctly understood sentences. We define that a sentence has been understood if the word sequence selected by the parser and refined by the feedback verification procedure (if applied) is equal to the uttered sentence or differs from it only for short function words that are not essential for understanding. The *failure* rate is the percentage of sentences for which no result has been obtained by the parser within the real-time imposed constraints. The *misunderstood* case arises when the selected solution is not the uttered one. Note that failures and misunderstandings have not the same effect: in fact in the case of failure the system is aware of not having understood the question and in a dialogue system the failure can activate a recovery action.

The parser has been implemented using the C language and presently runs on a Sun SparcStation 1. Experiments have been performed starting from 60C lattices produced by the recognition system from 60C different sentences uttered by 10 speakers and pertaining to the voice access to E-mail messages. The recognizer [Fissore *et al.* 1989] employs 305 context-dependent units, each of which is represented by a 3-state discrete density HMM. HMMs are trained with 8800 sentences uttered by 110 speakers. The speech signal, recorded from a PABX, is low-pass filtered at 5 kHz and sampled at 16 kHz. Features, computed every 10 ms time frame, include 12 cepstrum and 12 delta-cepstrum coefficients, plus energy and delta-energy.

	base		+best sent.	
	no ver.	verify	no ver.	verify
Understood	63.7%	69.2%	65.7%	72.2%
Failure	4.4%	5.7%	10.5%	11.3%
Misunderstood	31.5%	25.2%	23.8%	16.5%

Table 5: Experimental results on 600 test sentences

Table 5 reports the results for two kinds of config urations, each evaluated with the feedback verificatio procedure disactivated (no ver.) or activated (verify) The first configuration is the baseline one, in which lattice is analyzed as described in the above sections In the second configuration, we add into the lattice th best-scored sequence of words initially found by the rec ognizer as a side-effect of its analysis. This sequence though rarely correct, takes better into account inter word coarticulation and hence may contribute to th overall accuracy. In both configurations the maximur processing time is 5 seconds.

7 Conclusions

The effectiveness of a two-level architecture for con tinuous speech understanding has been demonstrate through a working system tested on several hundre sentences recorded through a PABX from 10 speakers The implementation of the linguistic processor stresse the design of efficient ways of representing language con straints into knowledge sources through the procedur of fusion, and the development of efficient score guide control algorithms to perform parsing. A verificatio procedure permits to increase understanding accurac by exploiting the capabilities of the recognition modul as a post-processor, able to acoustically reorder sen tences hypothesized by the linguistic processor and fin out words that were skipped by the parser. Analysi times as low as about twice real time are achieved on Sun SparcStation 1.

References

[Baggia *et al.* 1991a] P. Baggia, E. Gerbino, E. Giachin, and C. Rullent, "Efficient Representation of Linguistic Knowledge for Continuous Speech Understanding", *Proc. IJCAI 91*, Sydney, Australia, August 1991.

[Baggia *et al.* 1991b] P. Baggia, L. Fissore, E. Gerbino, E. Giachin, and C. Rullent, "Improving Speech Understanding Performance through Feedback Verification", *Proc. Eurospeech 91*, Genova, Italy, September 1991.

[Baggia *et al.* 1991c] P. Baggia, A. Ciaramella, D. Clementino, L. Fissore, E. Gerbino, E. Giachin, G. Micca, L. Nebbia, R. Pacifici, G. Pirani and C. Rullent, "A Man-Machine Dialogue System for Speech Access to E-Mail using Telephone: Implementation and First Results", *Proc. Eurospeech 91*, Genova, Italy, September 1991.

[Erman *et al.* 1980] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. Raj Reddy, "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty", *ACM Computing Survey* 12, 1980.

[Fillmore 1968] C. J. Fillmore, "The Case for Case", in Bach, Harris (eds.), *Universals in Linguistic Theory*, Holt, Rinehart, and Winston, New York, 1968.

[Fissore *et al.* 1988] L. Fissore, E. Giachin, P. Laface, G. Micca, R. Pieraccini, and C. Rullent, "Experimental Results on Large Vocabulary Continuous Speech Recognition and Understanding", *Proc. ICASSP 88*, New York, 1988.

[Fissore *et al.* 1989] L. Fissore, P. Laface, G. Micca, and R. Pieraccini, "Lexical Access to Large Vocabularies Speech Recognition", *IEEE Trans. ASSP*, Vol. 37, no. 8, Aug. 1989.

[Giachin and Rullent 1988] E. Giachin and C. Rullent, "Robust Parsing of Severely Corrupted Spoken Utterances", *Proc. COLING-88*, Budapest, 1988.

[Giachin and Rullent 1989] E. Giachin and C. Rullent, "A Parallel Parser for Spoken Natural Language", *Proc. IJCAI 89*, Detroit, August 1989.

[Giachin and Rullent 1990] E. Giachin and C. Rullent, "Linguistic Processing in a Speech Understanding System", *NATO Workshop on Speech Recognition and Understanding*", Cetraro, Italy, July 1990, R. de Mori and P. Laface, (eds.), Springer Verlag, 1991.

[Hayes *et al.* 1986] P. J. Hayes, A. G. Hauptmann, J. G. Carbonell, and M. Tomita, "Parsing Spoken Language: a Semantic Caseframe Approach", *Proc. COLING 86*, Bonn, 1986.

[Lee 1990] K.-F. Lee, "Context Dependent Phonetic Hidden Markov Models for Speaker Independent Continuous Speech Recognition", *IEEE Trans ASSP*, Vol. 38, no. 4, April 1990.

[Hays 1964] D. G. Hays, "Dependency Theory: a Formalism and Some Observations", Memorandum RM4087 P.R., The Rand Corporation, 1964.

[Poesio and Rullent 1987] M. Poesio and C. Rullent, "Modified Caseframe Parsing for Speech Understanding Systems", *Proc. IJCAI 87*, Milano, 1987.

[Sowa 1984] J. F. Sowa, *Conceptual Structures*, Addison Wesley, Reading (MA), 1984.

[Stock *et al.* 1989] O. Stock, R. Falcone, and P. Insinnamo, "Bidirectional Charts: a Potential Technique for Parsing Spoken Natural Language Sentences", *Computer, Speech, and Language*, 3(3), 1989.

[Tomita and Carbonell 1987] M. Tomita and J. G. Carbonell, "The Universal Parser Architecture for Knowledge-Based Machine Translation", *Proc. IJCAI 87*, Milano, 1987.

[Woods 1985] W. A. Woods, "Language Processing for Speech Understanding", in F. Fallside, W. A. Woods (eds.), *Computer Speech Processing*, Prentice Hall Int., London, UK, 1985.