Strategies to Tackle Problematic Input in a Spanish Dialogue System

Victoria Arranz, Núria Castell and **Montserrat Civit**
TALP Research Center
Universitat Politècnica de Catalunya
Campus Nord - C6, Jordi Girona, 1-3
08034 Barcelona, Spain
{varranz,castell,civit}@talp.upc.es

Abstract

This paper focuses on the strategies adopted to tackle problematic input and ease communication between modules in a Spanish railway information dialogue system for spontaneous speech. The paper describes the design and tuning considerations followed by the understanding and dialogue manager modules. The former needs to handle the problematic input received from the speech recogniser, while the latter needs to establish as agile and unambiguous a communication as possible.

1 Introduction

The nineties witnessed a boosting in spoken dialogue system development. The need for machines offering more natural and efficient ways to communicate reinforced the importance of such systems. Demand for "intelligent" and "human-like" applications has led researchers and developers in the field of Human Language Technologies to attain products such as *TRAINS* (Allen et al., 1996), *LIMSI ARISE* (Lamel et al., 1999), *CSELT* (Billi et al., 1998) and *Philips TABA* (Souvignier et al., 2000), in the train timetable information domain. Our system goes further than train scheduling and allows the user obtain a wide range of information about a train trip (e.g., prices, duration, onboard services) while providing a relatively user-friendly mixed-initiative communication exchange in Spanish.

A very important issue in such systems, and a main concern for the current work, is poor speech recognition and its side-effects on the system modules to handle it. (Nakano et al., 2000) claims to deal with them in the framework of unification grammar. Litman's (Litman and Pan, 2000) spoken dialogue system, *TOOT*, adopts a rather pragmatics-oriented approach: it automatically adapts its dialogue strategies so as to predict and respond to problematic input. (Zue and Glass, 2000), on the other hand, emphasises a more semantic-driven approach, as it is a more robust strategy. Our system follows this latter strategy and uses a semantic extractor that performs robust and flexible searches based on partial parsing, phrase structures, lexical key-words and morphological information (cf. section 5). Further, our system has also implemented a number of pragmatics-oriented strategies for the dialogue manager to overcome communication difficulties generated as a consequence of poor input (cf. section 6.2).

Our system is based on a study of real data. The collection and study of a real human-human corpus has preceded the collection and study of a human-machine, *Wizard-of-Oz* technique-based (Fraser and Gilbert, 1991), dialogue corpus. These data have been used for the development and training of the system modules and represent a valuable resource for further research as such.

2 System Overview

Our system architecture follows that of other such dialogue systems (Giachin and McGlashan, 1997). It opens with a speech recognition stage where the spoken utterance is translated into a word sequence. The speech recogniser used belongs to one of the project partners (Bonafonte et al., 1998) and it makes use of an acoustic model and a language model, the former being language-specific and the latter domain-specific and thus built for our task.

As expected when dealing with spontaneous speech, the recogniser needs to incorporate the treatment of extra-linguistic phenomena, such as pauses, coughs, background noise, etc. In spite of this, there remains a certain amount of noise and problematic input that will go on to the next module. This is, in particular, one of the issues to be addressed in the current paper (cf. section 3).

The transcription of the spoken utterance into text is sent to the understanding module. This module starts with a linguistic processing of the received input (see section 4) so as to extract its semantic content and formalise it inside *frames*, which summarise the intervention (cf. section 6.1).

A *frame* generated in the understanding module is sent to the dialogue manager (cf. section 6.2) which controls all the information exchanges between the user and the system, knowing when it holds enough information to carry out a query to the train company online database (DB), when it needs to consult the dialogue history, or ask the user for more information, etc. The dialogue manager will also generate *frames* to transmit the necessary information to the natural language generator. These *frames* are converted into sentences that the text-to-speech synthesiser will output as a spoken response to the user's query. Figure 1 in appendix A shows the general architecture of the system.

3 Problems to be Tackled

The understanding module receives the trascription of the spoken utterance generated by the recogniser. Unfortunately, as it is well known in spontaneous speech, erroneous transcriptions take place. On the one hand, the recogniser is not perfect and it generates transcription errors itself. On the other, dealing with spontaneous speech implies having to face problems such as disfluencies.

Three different types or recognition errors must be considered when designing the understanding module. The first one is **excess of information**, i.e., the recogniser adds words that do not belong to the user's utterance (see case 1 below). The second one is **erroneous recognition**, i.e, words detected by the recogniser do not match those really uttered by the user (see case 2). The third error type is **grammar errors**, i.e., orthographic transcriptions can produce grammatical errors, including changes in grammatical categories (see case 3).

`Case 1:`

- *U*: "*A qué hora llega a Barcelona*" ("What time does it get to Barcelona")
- *R*: "**Sí el tren** *a qué hora llega a Barcelona*" ("**Yes the train** what...")

`Case 2:`

- *U*: "*Cero uno*" ("Zero one")
- *R*: "*Qué precio uno*" ("What price one")

`Case 3:`

- Lack of preposition+determiner contractions: "*de el*" should be "*del*".
- Misuse of the indefinite article instead of the cardinal: "**un** *de julio*", instead of "**uno** *de julio*" (July 1).
- Erroneous orthographic transcriptions causing grammatical mistaggings: "*qué/que*" ("what/that"), "*e/he*" ("and/have"), etc.

These three error types can be handled at two different levels. At the recognition level, the tool should be tuned for the task domain. Also, the strategy of closing the entry channel, when necessary, should be considered. At a language processing level, our semantic information extractor can handle noisy input by focusing its extraction on specific syntactic phrases, lexical entries (word forms and lemmas), and even POS labels (cf. section 5).

Disfluencies are the main problem to be tackled when developing a dialogue system for spontaneous speech. Syntactic disfluencies (see case 4) have been considered when designing the semantic information extractor. Most cases are sorted out at this stage (cf.

section 5). Other disfluencies, such as lexical variations, pauses, noises,..., must be tackled by tuning the speech recogniser. Yet, the result of semantic extraction from the received input will also be implicitly confirmed by the dialogue manager before processing the query. Thus, there is an attempt to get round any misrecognitions that keeps the user-system communication open until (now or later) the system succeeds.

`Case 4:`

- *U*: "*el* **este** *primero que vaya a Bilbao y que vuelva de Bilbao a Barcelona* **el primero que vuelva de Bilbao a Barcelona**"

The following sections describe the tool tuning and strategies adopted so as to a) tackle all those language processing problems encountered by the language understanding and dialogue manager modules, and b) ease the communication and information exchange between the modules, addressing issues such as information contradictions and ambiguity.

4 Tuning NLP Tools

This section describes the tuning of the morphological analyser *MACO+* (*Morphological Analyzer Corpus Oriented*) (Carmona et al., 1998) and the shallow parser *TACAT* (Castellón et al., 1998) to our dialogue system. Both are robust and wide-coverage tools. The adaptation of *MACO+* has consisted in reducing the lexicon and treating proper names. *TACAT* has had its grammar adapted so as to handle dialogue input better.

4.1 Morphological Analyser

The detailed study of the corpora developed for the current project and of the train company database has lead us to refine the task lexicon: it has provided us with all necessary city and station proper names, as well as guided us in the reduction of the lexicon to the most-frequent domain words. Proper names have also been labelled in a specific manner so as to ease their recognition at a later stage. In what regards the frequency study, it has been performed automatically and later revised by hand in order to add missing forms that are considered relevant for the task.

The total number of word forms used by our system is 1,799 (as opposed to the 900,000 of the general lexicon), including 160 proper nouns. All tests carried out with the task lexicon have given the expected results, i.e., all forms have been assigned the correct label.

The final set of words currently in use by the system is made up of 297 closed-category words and 1,502 open-category ones. The former enclose determiners, pronouns, prepositions and conjunctions. The latter are nouns, adjectives, adverbs and verbs.

Besides reducing the number of forms, we have also reduced ambiguity (the number of possible interpretations for each form), in agreement with the specificity of the domain. Table 1 in appendix A reflects this reduction: the left-hand side shows the relation between the number of words and labels before the manual disambiguation process. The right-hand side shows the figures after the process.

As it can be seen, tuning both lexicon and the interpretation of each word to the task domain represents a very important reduction in ambiguity.

Once this tuning has been done, the result obtained by *MACO+* after analysing the sample sentence *"Me puedes decir horarios de trenes de Barcelona a Bilbao?"*[1] becomes:

```
Me yo PP1CS000
puedes poder VMIP2S0
* decir decir VMN0000
* horarios horario NCMP000 horario AQ0MP00
* de de SPS00
trenes tren NCMP000
* de de SPS00
Barcelona BARCE NP000C0
* a a SPS00
Bilbao 13200 NP000C0
? ? Fit
```

where only *horarios* is ambiguous (as opposed to the 5 resulting when using the general purpose analyser, here marked with an asterisk). This helps our disambiguator *RELAX* (*Relaxation Labelling Based Tagger*) (Padró, 1997) considerably, easing its task and speeding it up.

[1] "Can you tell me the train timetable from Barcelona to Bilbao?"

4.2 Syntactic Analyser

The syntactic analysis is performed with *TACAT*, a chart-based analyser making use of a chunk grammar that does not express any dependency relation between the constituents (Castellón et al., 1998). Every user turn is considered a sentence. The grammar is context-free since it was originally designed for non-restricted domain language and later adapted for the present application. This adaptation has consisted in a) taking certain analysis rules out, such as those for verbal periphrases, and mostly b) adding domain-specific rules, in particular those to detect days of the week, dates, hours and minutes. Some examples are[2]:

```
DAY-NP:
sn-dia ==> espec-ms, dia-set.
sn-dia ==> espec-ms, dia-set, numer-mp.
sn-dia ==> espec-ms, dia-set, data.

Day of the week:
dia-set ==> ncmn000(viernes).
dia-set ==> ncms000(sabado).
dia-set ==> ncms000(domingo).

MONTH-PP:
sn ==> mccp00, sp-mes.
sp-mes ==> sps00(de), ncms000(enero).
sp-mes ==> sps00(de), ncms000(febrero).

HOUR-NP:
sn-h ==> j-fp, grup-nom-h.
grup-nom-h ==> numer-fp, n-fp(horas).
```

This analyser takes as input the output of the morphological analyser once it has been processed by the disambiguator. Then it generates syntactic trees as the following:

```
[{pos=>S }
[{pos=>abre-interrog}
 [{pos=>Fia  ,forma=>"",lema=>""}]]
[{pos=>patons}
 [{pos=>pp1cso00,forma=>"Me",lema=>"yo"}]]
[{pos=>grup-verb}
 [{pos=>vmip2s0,forma=>"puedes",lema=>"poder"}]]
 [{pos=>infinitiu}
  [{pos=>vmn0000,forma=>"decir",lema=>"decir"}]]
[{pos=>sn}
 [{pos=>ncmp000,forma=>"horarios",lema=>"horario"}]]
[{pos=>grup-sp}
 [{pos=>sps00,forma=>"de",lema=>"de"}]
 [{pos=>sn}
  [{pos=>ncmp000,forma=>"trenes",lema=>"tren"}]]]
[{pos=>grup-sp}
 [{pos=>sps00,forma=>"de",lema=>"de"}]
 [{pos=>sn}
  [{pos=>np000c0,forma=>"Barcelona",lema=>"BARCE"}]]]
[{pos=>grup-sp}
 [{pos=>sps00,forma=>"a",lema=>"a"}]
 [{pos=>sn}
  [{pos=>np000c0,forma=>"Bilbao",lema=>"13200"}]]]
[{pos=>cierra-interrog}
 [{pos=>Fit,forma=>"?",lema=>"?"}]]]
```

Besides retaining its tree structure, this format contains POS tags, word forms and lemmas and becomes the input for the semantic information extractor.

5 Information Extraction and the PRE+ Environment

The semantic information to be generated for the dialogue manager (section 6.2) is based on the concept of *frame* (Minker et al., 1997), which can be easily translated into a DB query. A *frame* can have two different types of information: *concepts*, which are the specific information the user is enquiring about, and *cases*, which are the restrictions to be applied to the *concepts*. For instance, *DepartureTime* is a *concept* which can have different *cases*, such as *DepartureCity*, *DestinationCity*, etc.

Information extraction is performed by means of the production rule environment *PRE+* (Turmo, 1999). *PRE+* works with rules whose model is: *condition* ==> *action*. *Conditions* establish the morphosyntactic context to be satisfied for *actions* to be applied. In order to establish those search contexts, the project corpora (see section 1) have been manually studied to determine the selection criteria that are to be implemented in the rules. For instance, it has been observed that, in Spanish, *DepartureCity* is always preceded by prepositions *de, desde* (from). Thus, any city name preceded by *de, desde* can (and must) be interpreted as the value for *case DepartureCity*. This is expressed in this rule:

```
(rule DepartureCity1
   ruleset DepartureCity
   priority 10
   score [0,_,1,0]
   control forever
   ending Postrule
   (InputSentence ^tree <+a>tree_matching(
      [{pos=>grup-sp}
        [{lema=> de|desde}]
        [{pos=> np000c0, forma=>?forma}]]))
  (consulta ^attributes +attr)
   ->
   (?_ := Print(DepartureCity,?forma))
   (?_ := REM(DepartureCity,X,+a))
   (+attrfi := NRPush(DepartureCity,?forma,+attr))
   (delete 2)
   (create consulta ^attributes +attrfi))
```

The main condition in this rule is the one

[2] *Viernes, sabado, domingo*: "Friday, Saturday, Sunday". *Enero, febrero*: "January, February".

starting with *InputSentence*. This condition specifies the type of syntactic element, *grup-sp* (PP), to be searched for by the rule. This element must be a node of the syntactic tree and must have two daughters: the first one must have one of the prepositions *de, desde* as lemma; its sister must be a city proper name (must have tag *np000c0*[3]). Since this is the name providing the necessary information, *DepartureCity*, this form must be kept (as indicated in *forma=>?forma*) to be later used in the action of the rule.

The actions of the previous rule are:

```
(?_ := Print(DepartureCity,?forma))
(?_ := REM(DepartureCity,X,+a))
(+attrfi := NRPush(DepartureCity,?forma,+attr))
(delete 2)
(create consulta ^attributes +attrfi)
```

Leaving aside internal elements related to the functioning of *PRE+*, line 3 from this action should be further explained. It is this particular line that extracts the information and creates the *case+value* that will be sent to the dialogue manager in the next *frame*.

The main advantage of this formalism is that it allows to search for semantic information in a very flexible variety of ways: searching for word forms, lemmas and POS tags. Regarding syntax, phrase structure can be specified, if relevant, but it can be left unspecified otherwise. On the other hand, should the search need to determine the syntactic environment further than in our sample rule, it is possible to do so. For instance, the condition can be more restricted to specify either the type of sister nodes our initial *grup-sp* must have, or how many daughter (and/or granddaughter) nodes it can have, etc.

A rule from the same ruleset that holds a more complex condition is the following:

```
 [{pos=>S}
   [{pos=>grup-sp}
      [{lema=> con}]
      [{lema=> salida}]]
   [{pos=>sn}
      [{pos=> np000c0, forma=>?forma}]]]))
```

In this particular case, the specification of syntactic structure starts at sentence level (S). S must have two daughters, a PP and an NP. The daughters of the PP must be exactly *con, salida* (with, departure), while its sister NP must have as nucleus a city name.

[3]Digit #6, for semantic case, contains **c**, marking thus a *city* name. The other option is **e**, for *station*.

5.1 Rule Organization

Rules are structured according to a hierarchy that is established *a priori* and that determines their application conditions. In the present application, such hierarchy is rather flat since all rules are placed in three strata. The mother ruleset is called **top** and its daughters are mostly domain-specific **rulesets**. Each defined **rule** belongs to a unique ruleset. The dependecy relation between rulesets is always expressed with an *isa*:

```
(ruleset top
  control one_cycle
  sort_proc standard_sort_proc
  sort_type static)

(ruleset DepartureCity
        isa top)
```

The *belongs-to* relation between rules and rulesets is expressed by means of the *ruleset* property:

```
(rule DepartureCity1
   ruleset DepartureCity   ...)
```

Given these relations, rules inherit their properties from the rulesets they belong to. Further, properties inside the ruleset taxonomy are propagated from **top** to **rulesets**.

5.2 Ruleset Control Elements

As it can be seen in the previous example, rulesets have several control properties associated. These properties direct the way to apply the rulesets and are the following:

- **control**: It indicates the number of times the rules of a ruleset can be evaluated for application. For instance, value *one_cycle* means that all rules within the ruleset are considered once. It might happen that no rule is applied and then no information is extracted.
- **sort_proc**: It indicates the order of application of the rules within the ruleset, based on the individual priority of each rule. For example, *standard_sort_proc* is the default value and establishes a decreasing priority order.

- **sort_type**: It indicates when rules must be sorted. This can be done during rule application time (*dynamic* value) or inside the ruleset (*static* value).

5.3 Top Level Control of Rulesets

Further to the intra-ruleset control conditions, there is a global control at an inter-ruleset level. This top level control manages the order of application of the rulesets and allows to handle complex or ambiguous situations, such as the following.

There are situations of potential rule overlapping at an inter-ruleset level. For instance, when attempting to extract information about dates, the rules for *DepartureDate* and *ArrivalDate* extraction might overlap due to the different context they occur in. That is, since the context of the latter is much more restricted than that of the former[4], it is necessary to evaluate first the rule for the latter or else all dates will be assigned to the *DepartureDate case.*

Bearing in mind the situations encountered in the studied corpora, the evaluation of ruleset application follows the sequential order shown below. Rule **top1** belongs to ruleset **top** and it has high priority. In fact, this is the first rule applied when the semantic information extractor starts. The action of this rule creates an ordered *concept* list that determines the order of evaluation for the rulesets. This is due to the fact that each *concept* is associated to one ruleset.

```
(rule top1
  ruleset top
  priority 1
  control one
  ->
  (create concepts ^list
    [CS_DepartureTime
    DepartureCity, DestinationCity,
    DepartureDestinationCity,
    DepartureStation, DestinationStation,
    ArrivalDate, ArrivalInterval,
    DepartureDate, DepartureInterval, Fi])
   (create consulta ^attributes [])
   (?_ := Tacat2WM()))
```

[4]The application corpora have shown that the *ArrivalDate* attribute is far less usual than *DepartureDate* and it mostly occurs with the verb *arrive* or the noun *arrival.*

5.4 Rule Control Elements

At a rule level, there are also some control properties. The initial part of the left-hand side of one *DepartureCity* rule is shown here:

```
(rule DepartureCity1
   ruleset DepartureCity
   priority 10
   score [0,_,1,0]
   control forever
   ending Postrule
   (InputSentence ...
```

The priority of a rule inside the ruleset is expressed by means of the **priority** property. Despite the fact that the **control** property is already established at a ruleset level, it is also possible to restrict the application of individual rules using the same property.

The most important rule property is that of **score**. Combining the values of its four parameters a large number of syntactic analysis problems can be tackled. These parameters permit to express a) the use of tree permutations, b) the maximum depth of the search level, c) the maximum number of leaves allowed in between terminal elements, and d) the maximum number of leaves allowed inside each element. For example, if the system user says: "Can I have the timetables for the Barcelona Bilbao trains?", parameter (c) will have a value 0, so as to allow us to extract the *DepartureDestinationCity* value, despite not having the usual prepositions ("from Barcelona to Bilbao") next to the city names.

6 Semantic and Pragmatic Frame Tuning

Although our system is based on a detailed study of the domain-specific corpora, there are pragmatic issues that need to be taken into account. Initially, *frames* were established based on the type of semantic information to be exchanged between the user and the system, such as querying about a train departure time, the price of a trip, etc. At present, and after integrating both our understanding and dialogue manager modules within the whole dialogue system, *frames* have been tuned to the needs of the information exchange protocols and the limitations of the modules.

Such tuning has affected both content and format in order to achieve a more robust communication a) between the user and the system, as well as b) between the modules themselves. Further, the changes applied have established a higher classification level within the already existing *frame* typology, simplifying the whole structure. This simplification has been carried out bearing in mind the speech acts considered for this particular task:

- **User Speech Acts**: *Query*, *Incomplete*, *Return Trip*, *Affirmation*, *Negation* and *Closing*.

These speech acts have been resumed to four different frames (cf. section 6.1).

- **System Speech Acts**: *Query to the User*, *Explicit Confirmation*, *Implicit Confirmation*, *Answer to Query*, *Lack of Understanding*, *Opening*, *Closing*, *Offer* and *Web Stalled*.

These speech acts have been summarised in seven different *frames*. Thus, the system only needs to handle 11 main *frames*, which contain a very rich variety of information in a very limited number of structures.

6.1 *Frames* in the Understanding Module

To start with, given that the semantic extraction is performed by means of a set of rules written in *PRE+* (cf. section 5) and that this environment is based on lexico-syntactic patterns, all reasoning behind the content of a *frame* and its corresponding labelling is carried out by the dialogue manager. This entails some implementation constraints for the understanding module:

- This module will only generate a set of *frames* carrying one of the following: a) precise queries to the system; b) *case* information that cannot be allocated by this module in any *frame* and that can only be tagged as *incomplete* (this might represent some information demanded by the system to complete a query), and c) answers to system queries that only hold an affirmation or a negation.
- *Query frames* will be marked with **Q** and those containing incomplete information with **I**. *Affirmation* and *negation frames* will be labelled **A** and **N**[5], respectively.
- *Frames* will contain either a generic label or a non-generic one, where the former has one information field and only refers to the type of *frame*. The latter allows to include further information fields and thus provide further specifications on a particular *frame*. For instance, **I** opens a generic-label *frame*, while **Q** initiates a non-generic one. A *query frame* does always specify what particular information the user is enquiring about and it looks as follows:

```
((Q_DepartureTime
    (DepartureCity Barcelona)
    (DestinationCity Bilbao)))
```

In order to illustrate the difference in the *frame*-opening label, a generic *incomplete frame* providing information about two attributes for a query looks like this:

```
((I
  (DepartureDate 10-05-2001)
  (DepartureInterval 13.00-21.00)))
```

The complete *frame* is encapsulated as an information pack by means of brackets. There are three different levels of brackets: a first one containing each *case/attribute* and its value; a second one enclosing all *cases* with their *frame* label, thus building up simple *frames*, and a third one enclosing all information relevant to that dialogue turn. The latter can contain a single *frame* or allow for the building of *frame sets*, as it happens in the query below:

```
((Q_DepartureTime
    (DepartureCity Barcelona)
    (DestinationCity Bilbao))
 (Q_Price
```

[5]The *closing* speech act is also covered by the *negation* one given that the former is always preceeded by a negation and the dialogue manager is aware of the type of question such negation is replying to. That is, this module keeps a dialogue history which allows it to know what the user's answer refers to exactly.

(DepartureCity Barcelona)
(DestinationCity Bilbao)))

This frame encapsulation makes the communication exchange easier for the awaiting modules (dialogue manager and natural language generator), since it establishes the whole information structure and thus informs the coming module whether it should be waiting for any further simple *frames* within a structure or not.

Further on *query frames*, the list of query labels considered for the current domain task is: *Q_DepartureTime*, *Q_DepartureTime-R*[6], *Q_ArrivalTime*, *Q_ArrivalTime-R*, *Q_Price*, *Q_DepartureStation*, *Q_DestinationStation*, *Q_TripDuration*, *Q_TrainType*, *Q_Services*, *Q_DepartureDate*, *Q_DepartureDate-R*, *Q_ArrivalDate* and *Q_ArrivalDate-R*.

6.2 *Frame* Strategies for the Dialogue Manager

The dialogue manager receives one of the four *frame* types described above. Its proceeding onto consulting the database is subject to having all necessary information for the query. In any case, the dialogue manager will generate one of seven possible *frames*. This output *frame* will be sent to the natural language generator which is the module to translate the *frames* into system turns. Regarding format, these *frames* follow the same criteria as that in section 6.1.

The content of the *frames* follows the semantic requirements imposed by the speech acts in section 6. However, the number of *frames* has been reduced in relation to that of speech acts, obtaining seven main *frames* out of nine speech acts. This has been done in order to simplify the number of main *frame* structures the natural language generator might receive. The reduction consists in generating a unique *courtesy frame* covering the *opening*, *closing* and *offer* speech acts, as well as a further *waiting* courtesy message. This message is sent to the user when the system is about to process the DB query, just after sending an implicit confirmation, and it politely asks the user to wait.

Out of the seven output *frames*, two have been particularly designed to handle problematic situations. These are *explicit confirmation* and *answer to query*, which are further detailed below. Last but not least, the *web stalled frame* should be also mentioned. This one is a kind of negation given by the system when it cannot access the DB on the URL of the train company. It aims to apologise to the user and explain that the information cannot be accessed on that precise moment. This *frame* looks like this:

((N_Web-Stalled))

6.2.1 *Explicit Confirmation Frame*

This *frame* tackles information contradictions received by the dialogue manager that need to be sorted out before launching a query to the DB. After checking the dialogue history, the dialogue manager might encounter that a same attribute within a *frame* holds two different values. These values might have been extracted from different user turns and one might be a correction or a misunderstanding. In this case, the dialogue manager will send an *explicit confirmation frame*, asking for a certain value to be confirmed. Below follows an example of this *frame*:

((ECN_DestinationCity
(OldValue Valencia)
(NewValue Palencia)))

This is an actual case encountered in one of our human-machine corpus dialogues. The dialogue manager received these two different destination cities for a one-way trip.

6.2.2 *Answer to Query Frame*

Answer to query is the other rather complex output. It will contain the information resulting from the query and to be provided to the user, expressed by means of one *frame* or a *frame set*. Since our dialogue manager can handle a large amount of information in this output, its characteristics are detailed below.

It can be a *frame set* and, as such, it can hold information about several services for the

[6] This **R** refers to the return trip. Including it as part of the query label saves us from having to create a specific *frame* for this speech act.

same query (eg. different train-timetables for the same trip) or different information for several queries (eg. providing timetables, prices, services, etc.). When dealing with a *frame set*, each simple *frame* contained in it carries its own label, knowing exactly what type of information is provided by each part. All *frame* labels contain a field label indicating which *concept* they are replying to. These *concept* field labels are similar to the query labels shown in section 6.1.

An example of *Answer to Query frame* for the system turn "There are five trains on Friday. The first one leaves at 07.30 and the last one at 19.30" is this:

((INFO_DepartureTime
(NumberOfTrains five)
(DepartureDate 10-05-2001))
(INFO_DepartureTime
(RelativeOrderingNumber first)
(DepartureTime 07.30))
(INFO_DepartureTime
(RelativeOrderingNumber last)
(DepartureTime 19.30)))

7 Conclusions

This paper describes the strategies adopted to handle problematic input and ease communication between modules in a Spanish railway information dialogue system for spontaneous speech. The strategies here presented are those concerning the understanding and dialogue manager modules in particular.

On the other hand, the tuning of NLP tools has helped reduce morphosyntactic ambiguity, thus lending a hand to our semantic information extraction environment, *PRE+*. This environment is highly flexible, allowing to perform searches as restricted or general as necessary, and by means of syntactic structures, lexical items and morphological elements (such as lemmas or POS tags).

On the other hand, the semantic *frames* for the task have also been designed so as to avoid ambiguity and store semantic information in a clear and easy-to-exchange manner.

At present, the system modules are running locally (in one machine), although we are converting the architecture into a distributed one, resembling client-server architectures.

8 Acknowledgments

This project is partially financed by the CICYT (TIC98-0423-C06) and by the CIRIT (1999SGR150). Montserrat Civit is a researcher funded by the Ministry of Education of the Spanish government.

References

J.F. Allen, B.W. Miller, E.K. Ringger, and T. Sikorski. 1996. A robust system for natural spoken dialogue. In *Proc. of ACL'96*.

R. Billi, R. Canavesio, and C. Rullent. 1998. Automation of telecom italia directory assistance service: Field trial results. In *Proc. of IVTTA*.

A. Bonafonte, J.B. Mariño, A. Nogueiras, and J.A. Rodríguez. 1998. Ramses: El sistema de reconocimiento del habla continua y gran vocabulario desarrollado por la upc. In *Proc. of VIII Jornadas de I+D en Telecomunicaciones*, Madrid.

J. Carmona, S. Cervell, L. Màrquez, M.A. Martí, L. Padró, R. Placer, Rodríguez, M. Taulé, and J. Turmo. 1998. An environment for morphosyntactic analysis and parsing of unrestricted spanish text. In *Proc. of the 1st Conference on Language Resources and Evaluation. LREC'98*, Granada.

I. Castellón, M. Civit, and J. Atserias. 1998. Syntactic parsing of unrestricted spanish text. In *Proc. of the 1st Conference on Language Resources and Evaluation. LREC'98*, Granada.

N.M. Fraser and G.N. Gilbert. 1991. Simulating speech systems. *Computer Speech and Language*, 5(1).

E. Giachin and S. McGlashan. 1997. Spoken language dialogue systems. In S. Young and G. Bloothooft, editors, *Corpus-Based Methods in Language and Speech Processing*. Kluwer Academic Publishers, Dordrecht.

L. Lamel, S. Rosset, J.L. Gauvain, and S. Bennacef. 1999. The limsi arise system for train travel information. In *Proc. of ICASSP'99*.

D.J. Litman and S. Pan. 2000. Predicting and adapting to poor speech recognition in a spoken dialogue system. In *Proc. of 7th National Conference on AI (AAAI-2000)*, Austin, Texas.

W. Minker, S. Bennacef, and J.L. Gauvain. 1997. A stochastic case frame for natural language understanding. In *Proc. of ICSLP'97*.

M. Nakano, N. Miyazaki, N. Yasuda, A. Sugiyama, J. Hirasawa, K. Dohsaka, and K. Aikawa. 2000. Wit: A toolkit for building robust and real-time spoken spoken dialogue systems. In *Proc. of 1st SIGdial Workshop on Discourse and Dialogue*, Hong Kong.

L. Padró. 1997. *A Hybrid Environment for Syntax-Semantic Tagging*. Ph.D. thesis, Universitat Politècnica de Catalunya, Barcelona.

V. Souvignier, A. Kellner, B. Rueber, H. Schramm, and F. Seide. 2000. The thoughtful elephant: Strategies for spoken dialogue systems. *IEEE Trans. Speech Audio Processing*, 8.

J. Turmo. 1999. Pre+: A production rule environment. Technical report, Universitat Politècnica de Catalunya, Barcelona. Working report LSI-99-5-T.

V.W. Zue and J.R. Glass. 2000. Conversational interfaces: Advances and challenges. *Proc. of the IEEE. Special Issue on Spoken Language Processing*, 88(8).

Appendix A

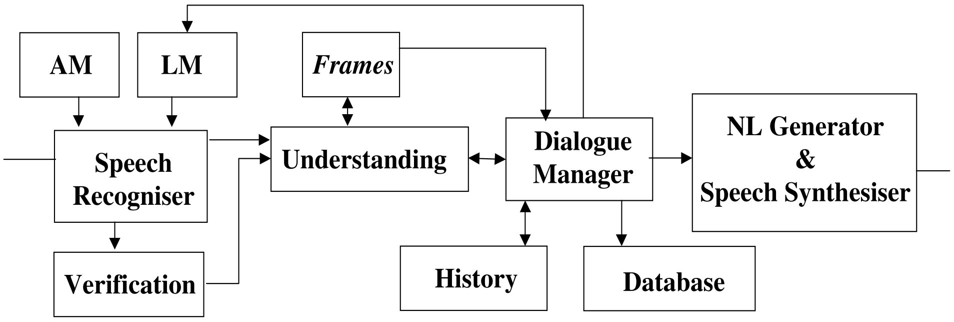

Figure 1: System Architecture

Table 1: Results from Ambiguity Reduction

Before		After	
words	labels	words	labels
1047	1	1175	1
469	2	493	2
197	3	116	3
53	4	10	4
26	5	5	5
5	6	-	-
2	7	-	-