A Framework for Processing Corrections in Task-Oriented Dialogues

Philip J. Hayes and Jairnc G. Carbonoll
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

Mundane discourse abounds with utterances referring to other utterances. These meta-language utterances appear with surprising frequency in task-oriented dialogues, such as those arising in the context of a natural language interface to an operating system. This paper identifies some simpler types of dialogue-level metalanguage utterance and provides a computational framework to process such phrases in the context of a case-frame parser exploiting strongly-typed domain semantics.

1. Introduction

ivteta-language utterances, i.e. utterances about other utterances in the discourse context, have received little attention in the computational linguistic literature. Yet, as a recent attempt [1] to create a taxonomy of discourse-level metalinguistic devices shows, such utterances are ubiquitous in all kinds of dialogues, including task-oriented man-machine dialogues through a natural language interface. Indeed, in a recent informal experiment with a simulated natural language command interface to a computer operating system, we found that one out of every twelve inputs was a metalanguage utterance of one type or another.

Meta language utterances fall into two primary categories, intra sentential and extra sentential (dialogue level) utterances. The former class may be divided into referential and interpretive meta-linguistic devices. For instance:

"Load, read and rewind the tape in that order."

is a typical referential intra-sentential meta language utterance of the kind that has received some attention in the linguistics literature [8, 6]. "In that order" refers to the surface structure sequencing of the lexical items denoting the commands. Interpretive metalanguage utterances make a statement about how another portion of that utterance should be interpreted, for instance:

"Metaphorically speaking, John's ideas are out of this world."

Discourse-level metalinguistic utterances refer to utterances or portions of utterances outside the scope of the immediate sentence. For instance, the third utterance in the dialogue fragment below is essentially a correction of a previous utterance.

USER:	Print gauss.rel on the diablo printer.
SYSTEM:	GAUSS.REL is not a printable file.
USER:	Oops, I meant gauss.for

This paper focuses on developing a process model for certain classes of dialogue-level metalanguage utterances, which occur in task-oriented dialogues with greater frequency than intra-sentential ones. With the exception of the analysis and classification of dialogue-level metalinguistic utterances in[1], there is little prior analysis of this phenomenon, although work in discourse focus [2, 10] and indirect speech acts [9, 7] is of direct relevance in analyzing the general phenomenon.

We currently have no general process-level theory for meta-language comprehension, but we have developed a framework for dealing with an important common subset of discourse-level meta language utterances: those that repair errors or misunderstandings that arose earlier in the dialogue, such as

"I meant to say transfer, not copy",

or

"Cancel that!".

in respose to an undesired action or a misinterpretation.

This class of utterances, which we call correction utterances, is particularly important for applied natural language because, at the present state of the art, we must expect a natural language interface to misint3rpret its user from time to time, and in any case, a friendly interface should always provide its user with an easy way to change his mind. In fact, in the informal experiment mentioned above, corrections accounted for more than half of all the metalanguage utterances.

The rest of this paper, then, will describe a framework for the recognition and interpretation of correction utterances in applied natural language systems, making clear the assumptions on which it is based that make it less than adequate as a model of correction in unrestricted dialogue. The techniques described here apply in the context of a case-frame parser with bottom-up constituent recognition and strong semantic typing, such as those described in [4, 3].

2. Types of Correction and their Interpretation

Suppose the following interaction occurs with a natural language interface to a computer operating system:

USER:	Transfer report.mss to the backup directory
SYSTEM:	REPORT.MSS inserted in the backup, and removed from the current directory.
USER:	I meant copy not transfer

The intention of the user's second input is clear: he wishes the system to behave as though his previous input had been "copy report.mss to the backup directory". To comply with this wish, the system has not only to execute the new command the user is specifying, but it also has first to undo the consequences of the original command that is being overridden. We call this kind of correction a replacement. The general form of a replacement is that the user specifies an earlier command and a change to be made to it, and the corresponding response from the system is to undo the effects of the specified command and to execute the changed version.

Note that while this response is the appropriate one from the computer system in this situation, this would not be the case in general. An instructor trying to teach the user how to use the system or a casual observer would react in quite different ways to

the same utterance. In general, as Carbonell [1] observes, the response to a meta-language utterance will be determined by many factors including the relative social roles of the dialogue participants and their relative capability for action. In our framework for interpreting correction utterances, we avoid having to deal with these complications by assuming the frozen social relation of master/slave between user and computer, and assuming that the computer is the only one with direct capability for action. It is these built-in assumptions that make the framework we are presenting somewhat less than a general solution of the interpretation of referential inter-sentential correction utterances, much less all metalanguage utterances. However, the simplifying assumptions do enable us to develop a computationally tractable model for a subset of meta-language constructions of considerable practical importance.

Means of identifying prior utterances and signalling desired replacement operations are not always as straightforward as in the example above. Consider, for instance:

USER: transfer report.mss to the backup directory
SYSTEM: REPORT.MSS inserted in the backup,
and removed from the current directory.
USER: I meant report.press

In this case, the command to be replaced is the same, but the user identifies it and the change to be made to it implicitly. A paraphrase of what he intended is "I meant to type 'report.press' instead of 'report.mss' in the previous command". It is very common to specify a replacement by mentioning a replacement object without mentioning the command in which the replacement is to be done, and often as in the example above without even mentioning the object to be replaced. In general, this can provide rather difficult reference problems requiring the use of intention modelling and focus tracking, but for applied natural language systems where the objects typically fall into one of a small number of distinct semantic types, the reference problem is considerably simplified. Our method for finding an object to be replaced where none is mentioned simply consists of selecting the last mentioned object whose semantic type is compatible with the replacement object, and selecting the utterance in which it participates as the one on which the replacement is to be done. This process is analogous to Henndrix's simple method of resolving ellipsis in the LADDER/LIFER data-base query interface [5].

A second broad class of correction utterances is that of *cancellations.* Examples are:

USER: Transfer report.mss to the backup directory
SYSTEM: REPORT.MSS inserted in the backup,
and removed from the current directory.
USER: No, cancel that transfer!

Just like replacements, cancellations refer to a previous command which the user wishes undone, the difference being that there is no new changed command to be executed in its place. In both cases, the reference may either be explicit or by mention of an object that was part of the command or (for replacements) by mention of an object of similar type.

3. Implementing the Framework

This section presents our plans for an implementation to deal with correction utterances in the context of a limited-domain natural language system. The algorithm we propose has four main steps:

1. Identify the input as a replacement or cancellation, and isolate the specification of what is to be replaced or cancelled.
2. Identify the earlier input upon which the replacement or cancellation operation will be performed. And, in the case of replacement, determine what part of the utterance should be replaced.
3. Determine whether the earlier input caused lasting effects, and undo them if possible and necessary[1].
4. In the case of a replacement, redo the earlier input after making the appropriate modifications.

Let us consider each of these steps in turn.

To identify meta-language utterances and isolate their important constituents, we plan to use the same case-frame based parsing procedure we have been developing for general use in restricted domain natural language interfaces [4,3]. Essentially the parser combines a top-down case frame instantiation process exploiting strongly typed semantic constituents in restricted domains, with a flexible bottom-up pattern matcher that recognizes individual constituents according to the semantic constraints attached to the slots of the case frames. The pattern matcher may recurse in its constituents, thereby providing at least the power of a context-free grammar.

For correction utterances, there would be two case frames, Replace and Cancel, which would be identified by sets of patterns instantiated by phrases such as "I meant to say", "It should have been" for Replace, and "Forget it", "I didn't intend to" for Cancel. Replace has two slots, Replacement and ToBeReplaced; the former is mandatory and unmarked; the latter is optional and marked by such phrases as "in place of" and "rather than". Cancel has only one optional unmarked slot, ToBeCancelled.

The parsing process would involve translating inputs with phrases that identified them as one of the two types of correction utterances into instantiated versions of these case frames, thus "I meant edit rather than delete" would result in

```
[CaseFrame: Replace
 Replacement: "edit"
 ToBeReplaced: "delete"]
```

while "Forget it!" would result in

```
[CaseFrame: Cancel
 ToBeCancelled: UNSPECIFIED-REFERENT]
```

The details of the notation are not important here, just the observation that the relevant parts of the input are being isolated.

After parsing, the next step is to identify which prior input is being replaced or cancelled, and in the case of replacement to decide how it should be modified. The procedure is different depending on whether the ToBeReplaced (ToBeCancelled) slot of the input parse is specified. If this slot is specified, then the previous input being modified is taken to be the last one in which the slot filler was mentioned, and for a Replace, the input is modified by replacing ToBeReplaced by Replacement. Thus for "I meant foo.bar rather than bar.foo", the last operation involving bar.foo would be replaced by one with foo.bar substituted in place of bar.foo. While for "I didn't mean to delete it", the last delete operation would be indicated. If, on the other hand, the ToBeReplaced (ToBeCancelled) slot is not specified, there are several possible methods of identifying the input to be modified:

- If it is a Replace, take the most recent input involving an object of the same semantic type as the Replacement slot, which must always be specified, and substitute the Replacement for that object. So in "I meant to say

[1] If the user wanted a different directory typed and has so specified via his meta utterance, no 'undoing' operation is required. However, if he wanted a different file copied, the new copy of the file should be deleted in order to undo the consequences of the previous action.

foo.bar", the Replacement is foo.bar, a file, and so the input to be modified is the last input mentioning a file, and the modification is to substitute foo.bar for that file.

- If there is a recent error message, modify the input that generated the error message, replacing the object of corresponding semantic type in the parse of the input utterance that generated the error. So, if "edit bar.foo" generates the error message "Non-Existent File: bar.foo", an input of "I meant to say foo.bar" would modify the edit operation by replacing the offending object, bar.foo, with the Replacement filler, foo.bar.
- Otherwise, modify the last input. In a case like "I didn't mean that", there seems little other choice.

After identifying and. if appropriate, modifying the relevant previous input, the next step is to undo any lasting effects of the action taken. Just what this means will vary depending on the operation specified by the input, and we will maintain a table of inverse operations, so that determination of the appropriate inverse will be straightforward. There are three main classes:

c There is no inverse and there are no harmful side-effects, e.g. type (a file on the terminal).

- There is a direct inverse, e.g. copy (a file) would have deleting the copy as its inverse. In this case, there would also be a mapping specifying how the arguments of the inverse operation would correspond to those of the original operation.
- The operation had significant, lasting side-effects that cannot be undone, e.g. list (a file on the line printer). In this case, the only thing that can be done is to tell the user the sad news, and ask if he wants to proceed with the modified operation anyway.

The final step in the overall algorithm applies only to replacements, and not to cancellations. It simply consists of executing the modified operation that was constructed in the second step. Because of the heuristic nature of the entire procedure, this execution of the modified operation and/or the undoing of the consequences of the modified operation should always for safety's sake be preceded by a request for the user's approval.

4. Problems

The procedure described in the previous section will handle as it stands most of the replacements and cancellations that we have encountered. Nevertheless, there are still many loose ends to be tied down and some problems to be solved, such as:

- Deep semantics of actions: A few instances of meta-language utterances require a much deeper understanding of non-linguistic actions and reasoning about their consequences than our present systems can handle. For instance, consider the following dialogue fragment:

USER: List gauss.for in the diablo printer.

SYSTEM: GAUSS.FOR queued for printing.

USER: Is GAUSS.FOR being printed?

SYSTEM: GAUSS.FOR is 25th on the queue.

USER: Forget it then!

The action that must be 'forgotten' is printing the file, not the last request for information about the status of the print queue. Clearly, knowledge of pending actions with irreversible consequences versus completed actions without physical consequences is necessary to resolve this example.

- Scope of the Undoing: If the user has performed several consecutive operations on file foo.bar, and says, "No, I meant bar.foo", he probably intends the whole sequence of operations to be undone and repeated for bar.foo, rather than just the last one. While this would be relatively straightforward modification to the existing scheme, there are more difficult cases. Whai happens, for instance, if all the operations on foo.bar are not consecutive?
- Undoing versus Repetition: If the user says "Use foo.bar instead of bar.foo", does he mean that the last operation on bar.foo should be undone, or merely repeated using foo.bar in place of bar.foo? It is clear that a system at the level we are proposing cannot hope to make the appropriate selection in a consistent manner, where 'appropriate' is defined to be the choice made by a human in the same situation. However, if all the alternate.s in an ambiguity are recognized and represented, user queries to resolve the interpretation problems could be generated in a focused manner [4].
- Level of ellipsis: With such a facility, a user might be tempted to use ellipsis at a level below the normal lexical level of the parser. For instance, he might follow "delete foo.mss" with "No, I meant .press", meaning undelete foo.mss and delete foo.press. Here .press may well be a unit smaller than the lexical items the system is capable of handling. He might even say, "No, I meant the press file", the proper processing of which would entail the system knowing that files with extensions '.mss' are often processed into files with the same name, but extension '.press', by a certain text formatting program, and that the latter kind of files are called press files.

5. References

1. Carbonell, J. G., "Beyond Speech Acts: Meta-Language Utterances, Social Roles, and Goal Hierarchies," *Preprints of the Workshop on Discourse Processes,* Marseilles, France, 1982.
2. Grosz, B. J., *The Representation and Use of Focus in Dialogue Understanding,* PhD dissertation, University of California at Berkeley, 1977, SRI Tech. Note 151.
3. Hayos, P. J., and Carbonell, J.G., "Multi-Strategy Construction Specific Parsing for Flexible Data Base Query and Update," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence,* August 1981, pp. 432-439.
4. Hayes, P.J. and Carbonell, J. G., "Multi-Strategy Parsing and its Role in Robust Man-Machine Communication," Tech. report CMU-CS 81 -118, Carnegie-Mellon University, Computer Science Department, May 1981.
5. Hendrix, G. G., Sacerdoti, E. D. and Slocum, J., "Developing a Natural Language Interface to Complex Data," Tech. report Artificial Intelligence Center., SRI International, 1976.
6. Joshi, A. K., "Use (or Abuse) of Metalinguistic Devices", Unpublished Manuscript.
7. Perrault, C. R., Allen, J. F. and Cohen, P. R., "Speech Acts as a Basis for Understanding Dialog Coherence," *Proceedings of the Second Conference on Theoretical Issues in Natural Language Processing,* 1978.
3 Ross, J. R., "Metaanaphora," *Linguistic Inquiry,* 1970.
9. Searle, J. R., "Indirect Speech Acts," in *Syntax and Semantics, Volume 3: Speech Acts,* P. Cole and J. L. Morgan, eds., New York: Academic Press, 1975.
10. Sidner, C. L., *Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse,* PhD dissertation, MIT, 1979, AI-TR 537.