A Method for Controlling the Production of Specifications in Natural Language

BENJAMIN MACIAS AND STEPHEN G. PULMAN*

University of Cambridge Computer Laboratory, New Museums Site, Pembroke St, Cambridge CB2 3QG, UK

Email: sgp@cl.cam.ac.uk

Before a system can be formally defined, it is common to write a specification in a natural language as the basis for the formal definition. Natural languages are not well suited for this task; documentation written in a natural language is often ambiguous and imprecise. Inspection of real documentation also reveals that, without special training, most writers do not produce concise, clear or consistent statements. We present here an experimental interface to a general-purpose natural-language processing system designed to control the writing of specification statements in a natural language. The interface is designed to reduce the degree of imprecision and ambiguity in the natural-language statements, as well as contributing to the writing of shorter statements, with clear structure and punctuation. We compare our approach to similar work in this area.

1. INTRODUCTION

It is generally accepted that the initial phase in the production of software must be a careful description of the system to be developed [1]. Any such specification should ideally be carried out in a formal language, but it is almost always the case that a previous specification in a natural language must be written. This is because formal-specification languages are often intelligible only to a small group of people, and formal specifications must thus be translated into a language that both client and developers can understand [2]. Several techniques have been advanced to assist non-specialists in the understanding of formal specifications, but natural-language specifications will continue to be used for the foreseeable future.

The use of natural language in this context is not without problems. It is well known that natural languages lack the properties we associate with those languages used for formal specification tasks, namely well-defined syntax and semantics. In order to use natural languages for specification purposes, we must therefore carefully consider how to do it, in order to reduce the risk of producing deficient specifications. It has been noted in the literature that the requirements specification phase, particularly the production of natural-language specifications, is one of the weakest links in the process of formally developing a system (e.g. [3–5]).

We describe in this paper a system consisting of a window-based interface to a natural-language processing system designed to write specifications in a controlled manner. Our primary goal is to provide a tool that can be used for this purpose in any setting. We assume that the typical users will be those with enough technical knowledge of the system to be specified, but not necessarily conversant with formal specification methods. Before we describe in more detail our system, we review the most salient characteristics found in some specification documents, and the problems they present.[1]

The fundamental problem of natural-language processing is the great amount of ambiguity at every linguistic level: morphological, syntactic, semantic and pragmatic. As expected, all these forms of ambiguity are to be found in natural-language specifications. We will not touch on them here, but will restrict ourselves to some overall characteristics of the documentation.[2]

The study of some of the documentation at our disposal revealed the following facts. First, we found a high average sentence length in the specifications. For example, one first case study has an average sentence length of 27.42 words; a sample of another study found an average length of 31.24 words. Treatment of sentences of such length is, without further restriction, beyond the state of the art of current natural-language processing. Sentences of such length can also be considered stylistically poor, and can lead to serious problems in comprehension.[3]

*Also at SRI International Cambridge Computer Science Research Centre.

[1]The work reported here has been sponsored by Engineering and Physical Sciences Research Council and the UK Department of Trade and Industry under the Department's Safety Critical Systems Advanced Technology Programme (SCSATP), project no. IED4/1/9001: A Method for Object Re-use in Safety-critical Environments. The source documents at our disposal were provided by our partners in the project: Lloyds Register, Ultra Electronics, Transmitton, West Middlesex Hospital, and British Aerospace Airbus Ltd.

[2]The pervasive ambiguity of language is the central problem of computational linguistics, and a great deal of the literature in the subject deals with it. [6] and [7] are introductions to computational linguistics; [8] discuss issues of ambiguity in the context of requirements specifications.

[3]This view is shared by some standards. For example, the AECMA Controlled English standard restricts writers to sentences of a maximum of twenty words.

Second, we found that punctuation is used in an extremely irregular manner, either because one writer does not use the same marks consistently, or because different writers have different intuitions of how to use them. This result suggests that punctuation marks should be incorporated into the grammar of the system sparingly, and writers should be discouraged from using punctuation marks whenever possible. Free use of punctuation, together with unlimited sentence length, also also to encourage writers to type obscure paragraph-long sentences. A typical example is:[4]

> If selected to automatic at MCC1 for keyboard control at the supervisory computer, air compressors can be individually stopped/started manually and the outlet valves, opened/closed manually from the supervisory computer keyboard, provided (in the case of the air compressor) the differential air pressure switch is calling for that compressor to run.

An important finding was that writers very often use only a few different supra–sentential structures to present the information. We will call.these structures presentational units. Among the most common in our documents, we have:

1. `if-then(-else)` statements: 'if the open signal from xv13 or xv14 is not received the duty pump will be inhibited from starting.'
2. `when` statements: 'when the level controller is switched to local operation, the set point will be [as] manually set at the controller.'
3. before and after statements: 'before starting the transfer procedure, the operator answers a menu in order to define the loading process.'
4. Lists, which are one of the most common presentational devices (according to our sample, up to 30–40% of a specification):

'The wash sequence initiation procedure will:
a. Co-ordinate the filter wash requests.
b. Formulate a wash queue if necessary.
c. Start the wash sequence.'

The last example is representative of the issues raised by the use of these structures. As it stands, it underspecifies a procedure: we do not know whether the result of the sequence is one or several of the actions on the list, or all three actions. We do not know either whether the procedures will be carried out sequentially (i.e. one after another), or in any order. Documentation containing a large proportion of such lists is clearly going to be inadequate.

To summarize, freely written specifications suffer from sentences that are too long, use punctuation erratically and use presentational units that introduce underspecified statements. These observations led us to the basic idea behind our system: rather than letting users type statements without constraints, we built a window-based interface to a natural-language processing system. By choosing from a menu of preset structures, users select the kind of presentational unit they want. The set of presentational units required are defined by a grammar which automatically configures the relevant menu choices. The interface directly connects to a natural-language system capable of analysing many ordinary English sentences. With this design, we achieve several objectives: the system is domain independent, it controls the use of potentially ambiguous presentational units, it naturally limits the length of the English sentences, and it reduces the need for punctuation to a minimum. As we will try to show below, these features of the system contribute to a considerable reduction in ambiguity and underspecification.

2. A WINDOW-BASED INTERFACE

We present now the architecture of the interface. It has been designed to present an easy to use, controlled interface to the natural-language processing system at our disposal, CLARE. CLARE [10] – which includes the Core Language Engine [11] as a component – is a general purpose natural-language processing system developed at SRI International, Cambridge. It has been used to build database query applications [12], spoken language dialogue systems [13] and bidirectional machine translation prototypes [14]. In the configuration used here, CLARE carries out morphological, syntactic, semantic and some contextual analysis on an input sentence and then presents one or more possible logical forms representing the interpretation(s) of the sentence.

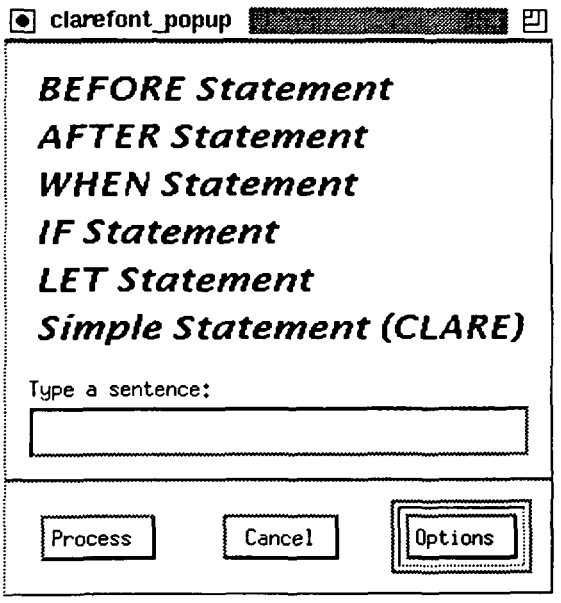
The user begins by selecting the kind of statement to be written, and recursively selects other statement types, or simply writes ordinary English sentences. Each sentence is passed on to CLARE and analysed. If the sentence has more than one logical analysis, the user is requested to select the one they intended, disambiguating the input. The source specification in English is stored together with its logical analysis, and can be subsequently be modified, extended or used to generate documentation. In principle, the logical analysis could be subject to further forms of processing: for example, it could be linked to appropriate parts of a formal specification, or cross-indexed to a diagram.

We introduce the basic operation of the interface through an example. Let us assume that the user wants to type in the following specification – based on an actual example: "Before starting the transfer procedure, the operator answers a menu to define the loading process (valves that will be closed or opened, the propane reservoir and the phases that will be used)."

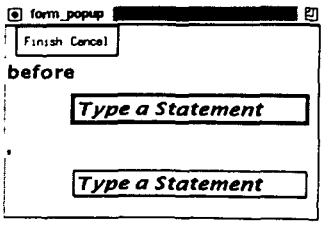
The system begins by displaying the options available to the user. After selecting the option to process a sentence, the system opens a window that displays the presentational units available, plus the option to type a

[4] [9], p. 84 notes the same phenomenon in the area of aviation maintenance manuals.

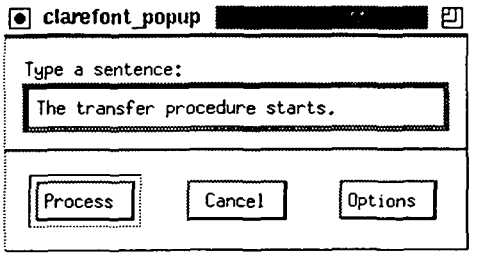
sentence directly:



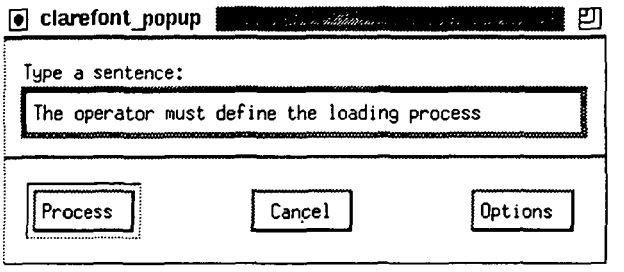
Let us begin with the first statement in the example: 'before starting the transfer procedure, the operator answers a menu to define the loading process'. This sentence type corresponds to one of the presentational units available, the `Before` statement. We can directly select the option from the menu, or type `Before` and hit the `Return` key. Because this is a predetermined statement, the system will retrieve its definition and structure. Selecting this option will produce a new window:



We choose now `Type a Statement` to input the first part of the `Before` statement. In the current implementation this will result in a new window where we can type the first English sentence:



If this sentence is in turn a complex one, then a new window is necessary. In this instance, the sentence is a simple one and the extra window a little redundant. After typing the sentence, it is processed by CLARE. The system finds that the sentence is grammatically correct, and that it has one logical analysis. It is then considered correct, and stored for future reference. We go back to the before window, and choose again to type an English statement with `Type a Statement`. As before, another window pops up, and we type:



The sentence is analysed and validated by the natural-language system; its logical analysis is saved. With this we finish. We have obtained a valid specification of the statement:

```
BEFORE
the transfer procedure starts,
the operator must define the loading process.
```

3. LISTS

Lists (conjunctions, disjunctions or sequences) are treated by an extension of the mechanism shown above. Assume now that the user wants to draft the second part of the specification, and that the list of actions in parentheses ('valves that will be ...') is to be interpreted as a sequential coordination. The process of specification is carried out through the same mechanisms as before: select a when statement from the menu, capture an English sentence, and then type in the conjunction. To do this, the use can either choose an option from the menu, or simply type:

```
The operator has defined which valves to open or
close and then
```

The sequence and then acts as a keyword to tell the interface that we have started a sequential conjunction; the system will loop until an input statement lacks a trailing and. We continue with the following sentences in the same fashion, until we obtain, instead of the plain English description, the controlled specification:

```
The operator has defined the loading process
WHEN
  the operator has defined which valves to
  open or close
  AND THEN
  the operator has defined the target
  propane reservoir
  AND THEN
  the operator has defined the phases
  that will be used.
```

To conclude, the process of choosing high-level presentational units from the menu, and typing English sentences for the natural-language system, has taken us from the freely written version: "Before starting the

transfer procedure, the operator answers a menu to define the loading process (valves that will be closed or opened, the propane reservoir and the phases that will be used)" to a controlled one:

```
BEFORE
The transfer procedure starts,
the operator must define the loading process.
  The operator has defined the loading process
WHEN
  the operator has defined which valves
  to open or close
  AND THEN
  the operator has defined the target
  propane reservoir
  AND THEN
  the operator has defined the phases
  that will be used.
```

This specific rendering of the free version is only one among many that we could have typed. Our purpose here is not to suggest this specific style of drafting, but merely to illustrate how it can be done using the interface. Other writers, with different styles, will produce alternative specifications. A range of variant forms is permitted by the system, within the limits imposed by grammatical coverage. As we hope this example suggests, the approach chosen allows for better specifications in a relatively unconstrained manner. We will return to the trade off between clarity and expressiveness in Section 9. We will only note here that in order to decide whether to constrain the style of specifications, the overall consideration will not be the complexity of the system being described (very complex systems can be described by simple specifications), but how best to combine clarity of expression with simplicity of description. If needed, the system can be tailored for specific applications.

4. DEFINING LOCAL IDENTIFIERS

Texts often use pronouns and definite noun phrases ("the ...") to refer to entities in the discourse. Unfortunately, they can also create uncertainties in the understanding of a text. For example, definite noun phrases can denote a specific object or objects in the discourse, can refer to some unspecified (and possibly non-existent) object fulfilling a description, or can be used generically to talk about the properties of a set of entities. Consider the following fragment:

An operator executes the starting procedure when
he opens the input valve, and then
he activates the pump attached to the input valve.
...
An operator executes the closing procedure when
he turns the pump attached to the input valve off,
and then he closes it.

Both sentences contain references to 'an operator' and 'the input valve'. In the absence of other information, it is not clear whether they refer to same entities or not. A similar situation applies to pronouns. The pronouns 'he' and 'it' need also to be resolved to obtain a complete interpretation of the specification.

To ameliorate this problem, we have introduced a LET facility. This feature makes it possible for the user to introduce global names, and associate them to entities that will be repeatedly used in the specification. In the example above, the same sentences captured using the system and the LET facility will look like:

```
LET V be the input valve.
LET P be the pump attached to V.
LET OP be the operator.
OP executes the starting procedure WHEN
  OP opens V
  AND THEN
  OP activates P.
OP executes the closing procedure WHEN
  OP turns P off
  AND THEN
  OP closes V.
```

There is an alternative – and more natural – way of achieving the same effect. This can be done by using a construct like this:

```
An operator, OP, executes the starting procedure WHEN
  OP opens the input valve, V,
  AND THEN
  OP activates the pump, P.
OP executes the closing procedure WHEN
  OP turns P off
  AND THEN
  OP closes V.
```

The first mention of an entity also introduces an identifier parenthetically, which can be used later on to establish reference to that entity. This is an extension of a mechanism already available in English. The result may read more naturally while still eliminating the possibility of this type of ambiguity.

Although the first technique works, it is not a particularly natural construct, linguistically speaking, and the resulting specifications look a little like expressions of a programming language. The tension between these two alternative ways of introducing identifiers points to the conflict that arises when simultaneously trying to achieve precision, conciseness, and intelligibility. Further work is required before we find how far we can stray from pure natural languages. Still, we believe that a certain amount of artificiality is justified by the mileage we get in terms of better specifications.

5. STORING, MODIFYING AND REUSING A SPECIFICATION

The process of capturing a specification statement is backed by a separate window that displays a text of the input at each moment of the process. The first use of this text window is to help users orientate themselves through the implicit tree created by selecting options on the main

input window. The text window for one such specification is:

This output is produced from an internal representation, which contains a detailed description of each statement, as well as its corresponding logical analysis. It can be used to produce various kinds of paraphrases according to the user's needs, or kept as an independent file for other kinds of processing. An example of an internal representation is:

```
spec(1,
     1,
     before(
        clare([id(op), executes,
               the, loading, procedure]),
        andthen( [clare([id(op), must,
                         specify, id(l)]),
                  clare([id(l), plus, id(s),
                         must, be, less,
                         than, id(max)])] ) ) ).
```

which corresponds to the specification:

```
BEFORE OP executes the loading procedure,
 OP must specify L
 AND THEN
 L plus S must be less than MAX
```

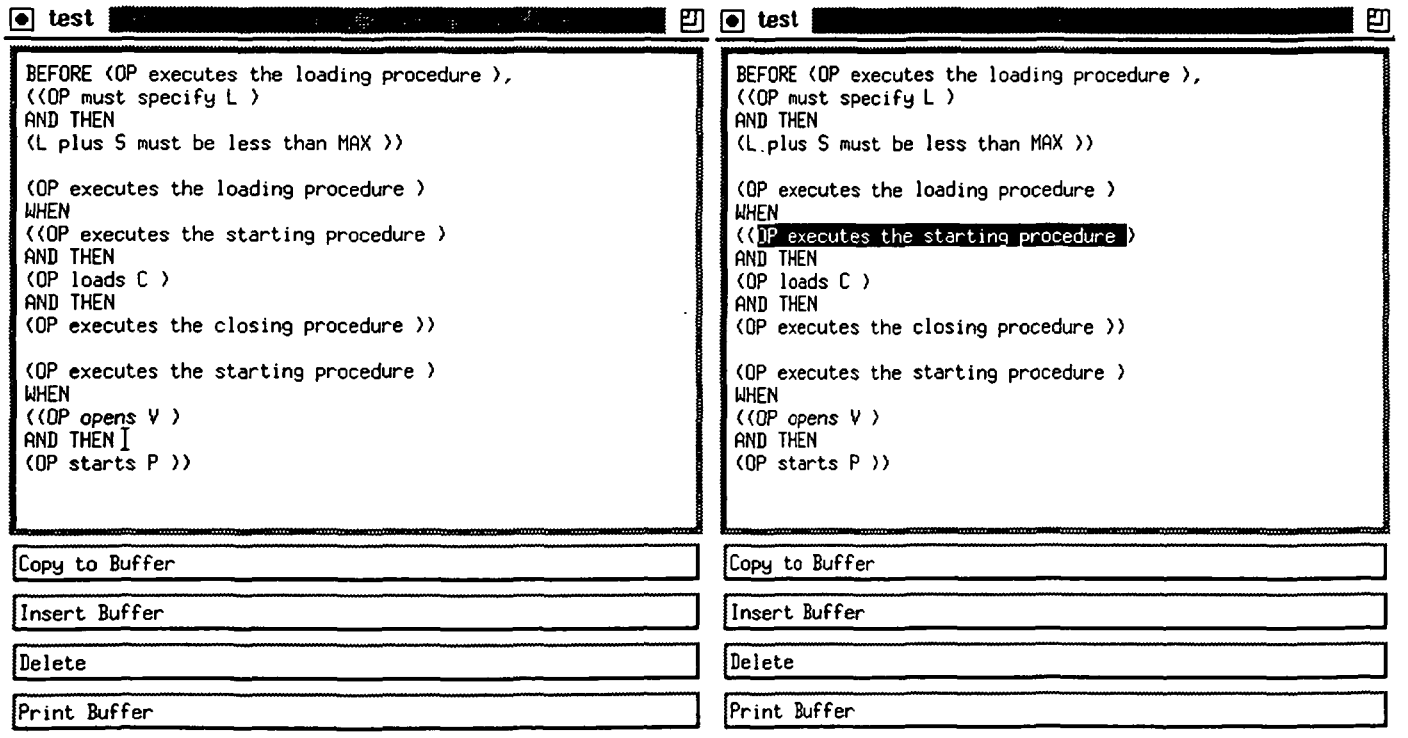
The second purpose of the text window is to copy, update or modify specific parts of a statement without having to rewrite it from scratch. This is done by first selecting from the text window a statement or substatement with the mouse:

auxiliary buffer for further manipulation. When a text is copied to the buffer, it can be added to the specification at any valid site. One such case occurs when a text has been deleted, creating a "hole" in the specification. Another possible insertion site is in a sequence of conjunctive statements. When there is such a site, and the buffer is not empty, the user can position the cursor there, and select insert. The program will insert there whatever is found in the buffer.

Both the text window and the auxiliary buffer are generated anew every time the specification is modified or added to. It would be possible for edits of this kind to introduce ambiguities into previously checked specifications, of course, and so reprocessing is necessary to confirm the final result.

6. AMBIGUOUS STATEMENTS AND PARAPHRASING

We have indicated above how to control for ambiguity at the level of presentational units. But to ensure that a complete statement is unambiguous, we must also check that each English sentence is unambiguous. We do this by adding a feature to the system that accepts a natural statement together with a specific interpretation. When CLARE processes a sentence and more than one logical analysis is found, the user is asked to choose among the various logical forms. The system then stores the original statement and its disambiguated logical form. This can be used for future reference, or to generate various paraphrases.



Once a text has been selected, it can be deleted or copied to a special buffer. If it is deleted, this action creates a copy of the statement which is put into an

Interactive disambiguation of this type is a large research topic in its own right. Linguistically naive users may not always notice that a sentence is ambiguous,

since contextual knowledge usually eliminates all but one linguistically possible interpretation, the others being unconsciously rejected as implausible. Furthermore, one cannot assume that the language used to represent the meanings of sentences (in our case, first-order logic enriched with a few higher-order constructs) will be familiar to the user.

We have chosen a simple temporary solution to this problem, presenting users with a "logicians' English" paraphrase of the logical forms representing the meanings of a sentence. For example, if the user types in the sentence:

```
The operator has stopped the process on the menu
```

the system will generate the following two (slightly simplified) analyses:

```
exists(A,operator(A) &
 exists(B,process(B) &
 exists(C,menu(C) & on(B,C) &
 exists(D, event(D) &
 exists(E, current_time(E) &
  precedes_in_time(D,E) &
  stop(D,A,B))))))
exists(A,operator(A) &
 exists(B,process(B) &
 exists(C,menu(C) &
 exists(D, event(D) &
 exists(E,
  current_time(E) & precedes_in_time(D,E) &
  stop(D,A,B) & on(E,C))))))
```

These analyses correspond to the interpretations in which the phrase "on the menu" modifies "process" or "stop" respectively (compare "the operator has read the message on the menu" and "the operator has positioned the mouse on the menu"). The system generates paraphrases of each, and asks the user to choose between them:

1. There is some operator OP, some process PRO (such that there is a menu ME, and PRO is on ME), and a past event E, such that E is an event of OP stopping PRO.
2. There is some menu ME, some operator OP, some process PRO, and some past event E, such that E is an event of OP stopping PRO, and E is on ME.

From these paraphrases, the user chooses the one that corresponds to the intended interpretation. The system then stores the desired interpretation with the input sentence, which can later be used to annotate the English source, or substitute it directly.

The CLARE system is bidirectional, and can generate sentences from a logical form. It would thus be possible to generate full English sentences from the logical form back to the user. However, we would need careful checking to ensure that we did not simply regenerate the original sentence, and we would also need to ensure that the paraphrase did not itself introduce some other unintended ambiguity.

7. EVALUATION OF THE SYSTEM

In order to evaluate the basic design, we asked a few subjects to try a pencil-and-paper version of the system. Although the current system contains some variants compared to the one used in the example, it is basically the same. The experiment thus allowed us to gauge the user's reaction to the system, and increase its functionality.

To evaluate the system, we asked the users to carry out the specification of some operations on a simple system. We gave them examples of the style of English that the system is intended to handle, but encouraged users to go beyond this if they felt hindered by it.

The example specification involved a container of some sort, with one input and one output. The input to the container was through a pump, and the container had a sensor to measure how full the container was. We attached three valves to the container as well, one for the input, one for the output, and one emergency valve. The users were asked to specify some simple operations, such as loading the container, based on the following informal model intended to be representative of a simple and already quite clear specification:

> The loading operation will be done as follows:
> 1. The operator will specify the amount of material to be loaded into the container.
> 2. The operator will open the input valve V1.
> 3. The operator will then start the pump attached to the input valve.
>
> After the process is over, the operator will turn the pump off, and he will close the input valve V1. The amount to be loaded will not exceed the capacity of the container. If this happens, the loading operation will stop, and the emergency signal will be activated.

The results of the experiment were the following. First, we asked the users whether the overall design made the specification easy to follow. Most thought that the resulting style was indeed easy to follow, although it was noted that it was perhaps too close to a programming language, and far from English. We also queried the users about the extent to which the system contributed to achieve the goals that we had set. Their response was that it had achieved those goals, but it was again noted that the resulting text was more difficult to follow than ordinary English.[5] It was the opinion of the users, especially those with specification experience, that it could be used in a realistic setting.

One comment that we took into account concerns the rigidity of the system. The version tried in the experiment did not include a facility to edit or change an ongoing specification. Several users pointed to the need of features to cut-and-paste, delete or modify a draft as one went along. As a result, we added the facility described above to correct this problem.

[5]This might of course reflect the shortcomings of an example we wrote, and not the system in general. As several users commented, some of our own drafts could have been done in a simpler and clearer way.

It was noted by the participants in the evaluation that the system needed to be supplemented by some kind of indication of the meaning as well as the form of the presentational units. For example, it was pointed out that the semantics of individual sentences such as 'OP executes the X procedure' is influenced by the presentational unit in which it occurs. If we have 'OP executes the X procedure WHEN...' then this statement usually defines what procedure X is, whereas 'OP executes the X procedure' in 'BEFORE OP executes the X procedure...' functions quite differently. In practice, the system should therefore be used in conjunction with a set of stylistic conventions (and presumably some explicit training) to give guidance to the writers on the intended meaning for each construct.

8. FURTHER WORK

The current prototype has been developed with as little customization of the NLP system as possible. We wanted to see how far we could go on the basis of already existing tools. Customization is restricted to the addition of necessary vocabulary items. While the system accurately processes all the examples used in this paper, its coverage is still not adequate for real applications. However, the CLARE system provides a wealth of tools for this kind of customization, given an adequate corpus of examples, and this extension of coverage is not a problem in principle.

Part of the process of customization requires the addition of domain knowledge in a form suitable for inference, for the purposes of disambiguation. The addition of such knowledge makes it possible to use the logical forms which are the output of the natural-language system for further types of processing.

Among those, the work reported here is part of larger effort to link natural-language statements and their corresponding formal specifications. In [15] we explored the idea of treating a formal specification of a system as a database, and used CLARE to pose queries in English about the – explicit and some implicit – properties of the specification. The system answers a query by producing a paraphrase in English of the formal contents of the database. Two prototypes were built: an interface to the Z specification [16] of the UNIX filestore by Morgan and Sufrin [17], and a safety-oriented RSL specification [18] of valves. For example, the first prototype answers questions such as *how do I read a file? or what are the conditions for reading a file?* by giving English-like paraphrases of the formal-specification statements that answer the questions.

We do this by axiomatizing in Prolog the properties of the domain entities and the specification language itself, linking logical analyses produced by the natural-language system and the underlying specification. The method requires a relatively small number of axioms (dozens rather than hundreds), and seems well-suited to domains that contain only a small number of entities. The techniques used here are similar to those used to build natural language interfaces to databases (see [19] for a review).

A more ambitious approach would be to attempt full-blown translation from natural-language statements to formal specifications. This is a considerably more difficult problem than merely obtaining a logical form of English statements. Although having a formal language (like Z) as target might simplify somewhat the problem, it probably resembles in complexity the problem of translating between two natural languages. Research in machine translation will have to advance significantly before realistic use can be made of a system built along these lines, specially where safety considerations are involved.

Natural-language paraphrases have been identified [2] as one of the ways in which users can improve their understanding of formal specifications, as well as the use of animation techniques (e.g. [20, 21]). The aim of the work we have just summarized can be seen as a bridge between them. By using natural-language techniques to enhance user comprehension of formal specifications, it can help in the adoption of formal methods in non-academic environments, and contribute to the industrial acceptance of formal methods [22].

9. RELATED WORK

The idea of building natural-language interfaces through menus was introduced by [23]. However, in the system described there the composition of basic sentences also had to be achieved by menu selection, one word at a time. It may be useful to adopt this technique, or some variant of it, if it is impossible to guarantee that the NL system will accurately analyse the basic sentences in any other way.

The use of natural-language techniques to help the process of specification writing has been explored by a few authors. [24] sketch a system that, among other capabilities, is able to build formal descriptions through natural-language dialogues. [25] describe a system to derive formal descriptions from natural-language specifications. [26] contains a description of a knowledge-based system that maintains a database of a software development project. This system uses a natural-language system interface. [27] have built a system for the specification of automatic teller machines. These systems all have the common characteristic that they are application specific, with the natural-language processing being hard-wired to the application in question. Our work has concentrated on using general purpose systems, on the assumption that customizing will eventually be less effort than starting over again for each new application.

Closer to our work is a system built by [28]. They describe a system designed to enforce the writing of specification statements in English according to the AECMA Simplified English (SE) standard. In particular,

this system controls various lexical (i.e use of authorized words), stylistic (e.g. maximum sentence length), and syntactic (it allows only one parse per sentence) criteria, according to the standard. However, no semantic processing is carried out. As such, the system operates as a sophisticated grammar checker.

Although the use of such predefined controlled languages can be of help in the process of writing specifications, it is not without shortcomings. First, statements written in a controlled language might be less clear than freely written ones. There is a trade off between striving for clarity and conciseness, while trying simultaneously to use a limited number of words. Without experimental evidence, it is not obvious whether controlled techniques deliver better documentation. Also, there is always the risk that controlled languages might be incomplete. In other words, how do we know that everything that needs to be said can be said in a controlled language? Excessive limitations can force the writer into producing statements that, although valid from the definition point of view, might not mean what the user wanted to write in the first place. We give an example based on the AECMA standard (taken from [27]. A writer might want to type the statement "do not touch any cable". Due to the limitations of the standard, they might end up by writing instead "do not touch the cable" or "do not touch all the cables". These are valid SE alternatives, but have different semantics from the original.

It may be that our own system can be criticized on similar grounds. However, our belief is that by using a general purpose NL system, we are in a good position to give specification writers as much freedom as is consistent with clarity. We would envisage that, in a situation where there was a continuing need for specifications around a particular subject area, a period of interaction with users would enable us to converge on a subset of general English that struck the right balance between clarity, freedom of expression and computational processability. With the system then customized to this subdomain, we would have all the advantages of the SE processors, along with the extra benefits arising from the fact that full semantic processing is being carried out.

Finally, we briefly mention the work done in computational linguistics in areas specifically related to documentation. A number of papers have been devoted specifically to the linguistic issues that manuals and other instructional texts present. [29] carried out a pioneering study of discourse phenomena in the context of task-oriented dialogues. The collection of papers by [30] contains studies of the linguistic features of languages used in specific domains. [31–34] have studied the use of rhetorical relations in manuals, with the goal of generating natural-sounding paraphrases. [35] have studied some semantic issues of linguistic expressions used in manuals. Some other authors such as [36–41] have looked at the question of how to generate natural-language expressions to paraphrase quantitative data, although without concern about the dangers of generating ambiguous expressions. An exception is [42], which contains other references to this line of work.

10. CONCLUSION

We have presented a system designed to help in the process of writing specifications in natural language. The system controls the writing of statements in English, reducing the length and syntactic complexity of sentences, and introducing some structure in the specification. The system also controls that the final statements are disambiguated, by forcing the user to choose between alternative analyses of a sentence. The supra-sentential presentational units that drive the system have Been obtained from real-life specifications. They seem to be very general, but if it is desired, the system can be easily tailored to other domains.

As mentioned in the introduction, one of the weakest links in the process of formally developing a system is the requirement specification phase, and in particular the problems associated to natural-language specifications. The use of formal techniques will hopefully become more common, specially for complex computer systems. The application of formal techniques will demand a better understanding of source specifications, both to improve the initial production of a system, as well as to support its maintainability and use.

REFERENCES

[1] Sommerville, I. (1989) *Software Engineering*, Addison-Wesley, Wokingham.
[2] Hall, A. (1990) Seven myths of formal methods, *IEEE Soft.*, 7(5), 11–19.
[3] Barroca, L.M. and McDermid, J.A. (1992) Formal methods: use and relevance for the development of safety-critical systems, *Comp. J.*, **35**(6), 579–592.
[4] Bloomfield, R.E. and Froome, P.K.D. (1986) Application of formal methods to the assessment of high integrity software, *IEEE Trans. Soft. Eng.*, **SE-12**(9), 988–993.
[5] Fraser, M. D., Kumar, K. and Vaishnavi, V.K. (1994) Strategies for incorporating formal specifications in software development, *Comm. ACM*, **37**(10), 74–86.
[6] Gazdar, G. and Mellish, C. (1989) *Natural Language Processing in Prolog: An Introduction to Computational Linguistics*, Addison-Wesley, Wokingham.
[7] Allen, J. (1987) *Natural Language Understanding*, Benjamin/Cummings, Menlo Park.
[8] Macias, B. and Pulman, S.G. (1993) Natural language processing for requirements specifications, in Redmill, F. and Anderson, T. (eds), *Safety-Critical Systems*, Chapman & Hall, London.
[9] Lehrberger, J. (1982) Automatic translation and the concept of sublanguage, in Kittredge, R. and Lehrberger, J. (eds), *Sublanguage: Studies of Language in Restricted Semantic Domains*, W. de Gruyter, Berlin.
[10] Alshawi, H., Carter, D., Crouch, R, Pulman, S. Rayner, M. and Smith, A. (1992) *CLARE, a Contextual Reasoning and Cooperative Response Framework for the Core*

Language Engine, Final Report, SRI International, December.
[11] Alshawi, H. (ed) (1992) *The Core Language Engine*, MIT Press, Cambridge, MA.
[12] Rayner, M. and Alshawi, H. (1992) Deriving database queries from logical forms by abductive definition expansion, in *Proc. 3rd Conf. App. Nat. Lang. Proc.*, 1–8.
[13] Lewin, I., Russell, M., Carter, D., Browning, S., Ponting, K. and Pulman, S. (1993) A speech-based route enquiry system built from general purpose components, *Proc. 3rd European Conf. on Speech Communication and Technology (EUROSPEECH 93)*, 2047–2050.
[14] Rayner M., Alshawi, H., Bretan, I. et al. (1993) A speech to speech translation system built from standard components, in *Human Language Technology, Proc. ARPA Workshop* 217–222.
[15] Macias, B. (1994) *Use of CLE/CLARE in Semi-automatic Translation of English and English-like Specifications*, MORSE Technical Report, Doc Id: MORSE/CU/BM/6/V2, June.
[16] Spivey, J. M. (1989) *The Z Notation: A Reference Manual*, Prentice Hall, New York.
[17] Morgan, C. and Sufrin, B. (1993) Specification of the Unix filing system, in Hayes, I. (ed.), *Specification Case Studies*, 2nd edn, Prentice Hall, New York.
[18] RAISE Language Group (1992) *The RAISE Specification Language*, Prentice Hall, New York.
[19] Copestake, A. and Spärck Jones, K. (1990) Natural language interfaces to databases. *Know. Eng. Rev.*, **5**, 225–249.
[20] West, M.M. and Eaglestone, B.M. (1992) Software development: two approaches to animation of Z specifications using Prolog, *Soft. Eng. J.*, **July**, 264–276.
[21] O'Neill, G. (1992) Automatic translation of VDM specifications into standard ML programs, *Comp. J.*, **35**(6), 623–624.
[22] Bowen, J. and Stavridou, V. (1993) The industrial take-up of formal methods in safety-critical and other areas: a perspective, in Woodcock, J.C.P. and Larsen, P.G. (eds), *FME'93: Industrial-Strength Formal Methods*, Springer, Berlin, pp 183–195.
[23] Tennant, H.R., Ross, K.M. Saenz, R.M., Thompson, C.W. and Miller, J.R. (1983) Menu-based natural language understanding *Proc Assoc. Comp. Ling.*, 151–158.
[24] Bergland, G.D., Krader, G.H., Smith, D.P. and Zislis, P.M. (1990) Improving the front end of the software-development process for large-scale systems, *AT&T Tech. J.*, **March-April**, 7–21.
[25] Miriyala, K. and Harandi, M.T. (1991) Automatic derivation of formal software specifications from informal descriptions, *IEEE Trans. Soft. Eng.*, **17**(10), 1126–1142.
[26] Devanbu, P., Brachman, R. J., Selfridge, P. G. and Ballard, B.W. (1991) LaSSIE: a knowledge-based software information system, *Comm. ACM*, **34**(5), 35–49.
[27] Fuchs, N.E., Hofmann, H. and Schwitter, R. (1994) *Specifying Logic Programs in Controlled Natural Language*, University of Zurich, Dept. Comp. Sci. Tech. Rep. 94.17.
[28] Hoard, J.E., Wojcik, R. and Holzhauser, K. (1992) An automated grammar and style checker for writers of simplified English, in O'Brian Holt, P. and Williams, N. (eds), *Computers and Writing: State of the Art*, Oxford, Intellect, pp 278–296.
[29] Grosz, B. (1978) Discourse knowledge, in Walker, D.E. (ed), *Understanding Spoken Language*, Elsevier North-Holland, New York, pp 229–344.
[30] Kittredge, R. and Lehrberger, J. (eds) (1982) *Sublanguage: Studies of Language in Restricted Semantic Domains*, W. de Gruyter, Berlin.
[31] Vander Linden, K. (1993) Generating effective instructions, *Proc. Ann. Conf. Cog. Sci. Soc.*, 1023–1028.
[32] Vander Linden, K. (1994) Generating precondition expressions in instructional text, *Proc. Assoc. Comp. Ling.*, 42–49.
[33] Rösner, D. and Stede, M. (1992) Customizing RST for the automatic production of technical manuals, in Dale R., Hovy, E., Rösner, D. and Stock, O. (eds), *Aspects of Automated Natural Language Generation*, Springer, Berlin, pp. 199–214.
[34] Peter, G. and Rösner, D. (1994) User-model-driven generation of instructions, *User Modeling and User-Adapted Interaction*, **3**, 289–319.
[35] Di Eugenio, B. and White, M. (1992) On the interpretation of natural language instructions, *Proc. Int. Conf. Comp. Ling. (COLING)*, 1147–1151.
[36] Kukich, K. (1983) The design of a knowledge-based text generator, *Proc. Assoc. Comp. Ling.*, 145–150.
[37] Rösner, D. (1987) SEMTEX: a text generator for German, in Kampen, G. (ed), *Natural Language Generation: New Results in Artificial Intelligence, Psychology and Linguistics*, Martinus Nijhoff, Dordrecht, pp. 133–148.
[38] Iordanskaja, L., Kittredge, R. and Polguere, A. (1991) Lexical selection and paraphrase in a meaning text generation model, in Paris C., Swartout, W. and Mann, W. C. (eds), *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, Kluwer, Dordrecht, pp. 293–312.
[39] Iordanskaja, L., Kim, M., Kittredge, R., Lavoie, B. and Polguere, A. (1992) Generation of extended bilingual statistical reports, *Proc. Int. Conf. Comp. Ling. (COLING)*, 1019–1023.
[40] Bourbeau, L., Carcagno, D, Goldberg, E. Kittredge, R. and Polguere, A. (1990) Bilingual generation of weather forcasts in an operations environments, *Proc. Int. Conf. Comp. Ling. (COLING6)*, 90–92.
[41] Robin, J. (1993) A revision-based generation architecture for reporting facts in their historical context, in Horacek, H. and Zock, M. (eds), *New Concepts in Natural Language Generation*, Frances Pinter, London.
[42] McKeown, K., Kukich, K. and Shaw, J. (1994) Practical issues in automatic documentation generation, *Proc Assoc. Comp. Ling.*, 7–14.