LEXICAL SEMANTICS IN HUMAN-COMPUTER COMMUNICATION

Jarrett Rosenberg

Xerox Office Systems Division
3333 Coyote Hill Road
Palo Alto, CA 94304 USA

ABSTRACT

Most linguistic studies of human-computer communication have focused on the issues of syntax and discourse structure. However, another interesting and important area is the lexical semantics of command languages. The names that users and system designers give the objects and actions of a computer system can greatly affect its usability, and the lexical issues involved are as complicated as those in natural languages. This paper presents an overview of the various studies of naming in computer systems, examining such issues as suggestiveness, memorability, descriptions of categories, and the use of non-words as names. A simple featural framework for the analysis of these phenomena is presented.

0. Introduction

Most research on the language used in human-computer communication has focused on issues of syntax and discourse; it is hoped that one day computers will understand a large subset of natural language, and the most obvious problems thus appear to be in parsing and understanding sequences of utterances. The constraints provided by the sublanguages used in current natural language interfaces provide a means for making these issues tractable. Until computers can easily understand these sublanguages, we must continue to use artificial command languages, although the increasing richness of these languages brings them closer and closer to being sublanguages themselves. This fact suggests that we might profitably view the command languages of computer systems as natural languages, having the same three levels of syntax, semantics, and pragmatics (perhaps also morpho-phonemics, if one considers the form in which the interaction takes place with the system: special keys, variant characters, etc.).

A particularly interesting and, till recently, neglected area of investigation is the lexical semantics of command languages. What the objects and actions of a system are called is not only practically important but also as theoretically interesting as the lexical phenomena of natural languages. In the field of natural language interfaces there has been some study of complex references, such as Appelt's (1983) work on planning referring expressions, and Finin's (1982) work on parsing complex noun phrases, but individual lexical items have not been treated in much detail. In contrast, the human factors research on command languages and user-interface design has looked at lexical semantics in great detail, though without much linguistic sophistication. In addition, much of this research is psycholinguistic rather than strictly linguistic in character, involving phenomena such as the learning and remembering of names as much as their semantic relations. Nevertheless, a linguistic analysis may shed some light on these psycholinguistic phenomena. In this paper I will present an overview of the kinds of research that have been done in this area and suggest a simple featural framework in which they may be placed.

1. Names for Actions

By far the greatest amount of research on lexical semantics in command languages has been done with names for actions. It is easy to find instances of commands whose names are cryptic or dangerously misleading (such as Unix's *cat* for displaying a file, and Tenex's *list* for printing), or ones which are quite unmemorable (as are most of those in the EMACS editor). Consequently, there have been a number of studies examining the suggestiveness of command names, their learnability and memorability, their compositional structure, and their interaction with the syntax of the command language.

Suggestiveness. In my own research (Rosenberg, 1982) I have looked at how the meaning of a command name in ordinary English may or may not suggest its meaning in a text editor. This process of suggestiveness may be viewed as a mapping from the semantics of the ordinary word to the semantics of the system action, in which the user, given the name of command, attempts to predict what it does. This situation is encountered most often when first learning a system, and in the use of menus. A few simple experiments showed that if one obtains sets of features for the names and actions, a straightforward calculation of their similarity can predict people's guesses of what particular command names denote.

Memorability. If we look at the converse mapping from actions to names, i.e., when, given a system action, one attempts to remember its name, we find a number of studies reporting similar results. Barnard *et al.* (1982) had subjects learn a set of either specific or general commands, and found that subjects learning the less distinctive, general names used a help menu of the commands and their definitions more often, were less confident in recalling the names, and were less able to recall the actions of the commands. Black and Moran (1982) found that high-frequency (and thus more general) words were less well

remembered than low-frequency ones, and so were more "discriminable" names (ones having a greater similarity to their corresponding actions). Scapin (1981) also found that general names like *select* and *read* were less well recalled than computer-oriented ones like *search* and *display*. Both Black and Moran (1982) and Landauer *et al.* (1983) found that users varied widely in the names they preferred to give to system actions, and that user-provided names tended to be more general and thus less memorable.

Congruence and hierarchicalness. Carroll (1982) has demonstrated two important properties of command name semantics: congruence and hierarchicalness. Two names are congruent if their relations are the same as those of the actions onto which they are mapped. Thus the inverse actions of adding and subtracting text are best named by a pair of inverses such as *insert* and *delete*. As might be expected, then, Carroll found that congruent names like *raise–lower* are easier to learn than non-congruent ones like *reach–down*.

Hierarchicalness has to do with the compositionality of semantic components and their surface realization. System actions may have common semantic components along with additional, distinguishing, ones (e.g., moving vs. copying, deleting a character vs. deleting a word). The degree of commonality may range from none (all actions are mutually disjoint) to total (all actions are vectors in some *n*-dimensional matrix). Furthermore, words or phrases naming such hierarchical actions may or may not have some of their semantic components realized on the surface; for example, while both *advance* and *move forward* may have the semantic features +MOVE and +FORWARD, only the latter has them realized on the surface. Thus, in hierarchical names the semantic components and their relationships are more readily perceived, thus enhancing their distinctiveness. Not surprisingly, Carroll has found that hierarchical names, such as *move forward–move backward*, are easier to learn than non-hierarchical synonyms such as *advance–retreat*. Similar results on the effect of hierarchical structuring are reported by Scapin (1982).

Names and the command language syntax. There are two obvious ways in which the choice of names for commands can interact with the syntax of the command language. The first involves selection restrictions associated with the name. For example, one usually *deletes* objects, but *stops* processes; thus one wouldn't normally expect a command named *delete* to take both files and process-identifiers as objects.

The second kind of interaction involves the syntactic frames associated with a word. For example, the sentence frame for *substitute* ("substitute *x* for *y*") requires that the new information be specified before the old, while the frame for *replace* ("replace *y* with *x*") is just the opposite. A name whose syntactic frame is inconsistent with the command language syntax will thus cause errors. It should be noted that Barnard *et al.* (1981) have shown that total syntactic consistency can override this constraint and allow users to avoid confusion, but their results may be due to the fact that the set of two-argument commands they studied always had one argument in common, thus encouraging a consistent placement. Landauer *et al.* (1983) found that using the same name for semantically similar but syntactically different commands created problems.

Non-words as names. Some systems use non-words such as special characters or icons as commands, either partly or entirely. Hemenway (1982) has shown that the issues involved in contructing sets of command icons are much the same as with verbal names. There are two basic types of non-words: those with some semantics (e.g., '?' or pictorial icons) and those with little or none (e.g., control characters or abstract icons). Non-words with some semantics behave much like words (so, for example, '?' is usually used as a name for a query command). Meaningless non-words must have some surface property such as their shape mapped onto their actions. For example, an abstract line-drawing icon in a graphics program (a "brush") might have its shape serve as an indicator of what kind of line it draws. Control characters are often mapped onto names for actions which begin with the same letter (e.g., CONTROL-F might mean "move the cursor *F*orward one character"). Similar considerations hold for the use of non-words to denote objects.

2. Names for Objects

In addition to studies of command names, there have been a number of interesting studies of how users (or system designers) denote objects. One version of this has been called the "Yellow Pages problem:" how does a user or a computer describe a given object in a given context?

Naming objects. Furnas *et al.* (1983) asked subjects to describe or name objects in various domains so that other people would be able to identify what they were talking about. The subjects were either to use key words or normal discourse. It was found that the average likelihood of any two people using the same main content word in describing the same object ranged from about 0.07 to 0.18 for the different domains studied.

Carroll (1982) studied how people named their files on an IBM CMS system (CMS filenames are limited to 18 characters and are thus usually abbreviated). Subjects gave him a list of their files along with a description of their contents, and from this, Carroll inferred what the "unabbreviated" filenames were. He found that 85 percent of the filenames used simple organizing paradigms, two of which involved the concepts of congruence and hierarchicalness discussed above.

Naming categories. Dumais and Landauer (1983) describe two major problems in naming and describing categories in computer systems. The first is that of inaccurate category names: a name for a category may not be very descriptive, or people's interpretation of it may differ radically. The second problem is that of inaccurate classification: categories may be fuzzy or overlapping, or there may be many different dimensions by which an object may be classified. Dumais and Landauer examined whether categories which are hard to describe could be better named simply by giving example of their members. They found that presenting three examples worked as well as using a description, or a description plus examples. In another study involving people's descriptions of objects (Dumais and Landauer, 1982) they found that their subjects' descriptions were often vague, and rarely used negations. The most common paradigm for

describing objects was to give a superordinate term followed by several of the item's distinctive features.

Deixis. The pragmatic issue of deixis should be mentioned, since some systems allow context-dependent references in some contexts such as history mechanisms. For example, in INTERLISP the variable *IT* refers to the value of the user's last evaluated top-level expression, but sometimes this interpretation does not map exactly onto the one the user has. Physical pointing devices such as the "mouse" allow deixis as a more natural way of denoting objects, actions, and properties in cases where it is difficult or tedious to indicate the referent by a referring expression.

There are, of course, many other aspects of the lexical semantics of command languages which cannot be covered here, such as abbreviations (Benbasat and Wand, 1984), automatic spelling correction of user inputs (Durham *et al.*, 1983), and generic names (Rosenberg and Moran, 1984).

3. A Featural Framework

While the above results are interesting, they are disappointing in two respects. To the designer of computer systems they are disappointing because it is not clear how they are related to each other: there are no general principles to use in deciding how to name commands or objects, or what similarities or tradeoffs there are among the different aspects of naming in computer systems. To the linguist or psycholinguist they are disappointing because there is no theory or analytic framework for describing what is happening. In my own work (Rosenberg, 1983) I have tried to formulate a simple featural framework in which to place these disparate results. My intention has been to develop a simple analysis which can be used in design, rather than a linguistic theory, but linguists will easily recognize its mixed parentage. At least a framework using semantic features has the advantage of simplicity, and can be converted into a more sophisticated theory if desired.

In such a featural approach the features of a name or action can be thought of as properties falling into four major classes:

- *Semantic* features are those elemental components of meaning usually treated in discussions of lexical semantics. For example, *insert* has the semantic feature + ADD.
- *Pragmatic* features are meaning components which are context dependent in some sense, involving phenomena such as deixis or presuppositions. For example, an anaphoric referent like *it* has some sort of pragmatic feature, however one wishes to describe it. It goes without saying that the distinction between semantic and pragmatic features is not a clear one, but for practical purposes that is not terribly important.
- *Syntactic* features are the sorts of selection restrictions, etc. which coordinate the lexical item into larger linguistic units such as entire commands. For example, *substitute* requires that the new object be specified before the old one.
- *Surface* features are perceptual properties such as sound or shape. The usefulness of including them in the analyis is seen in the discussion of non-words as names.

As Bolinger (1965) pointed out long ago, names and actions have a potentially infinite number of features, but in the restricted world of command languages we can consider them to have a finite, even relatively small number. Furthermore, only some features of a name or action are relevant at given time due to the particular contexts involved: the *task context* is that of the task the user is performing (e.g., text editing vs. database querying); the *name context* is that of the other names being used; and the *action context* is that of the other actions in the system. These three kinds of context emphasize some features of the names and actions and make others irrelevant.

Applying this framework to system naming, we can represent system actions and objects and their names as sets of features. The most important aspect of these feature representations is their *similarity* (or, conversely, their *distinctiveness*). This featural similarity has been formally defined in work by Tversky (1977, 1979).

Within these two domains of names and actions (or objects), *distinctiveness* is of primary importance, since it prevents confusion. *Between* the two domains, *similarity* is of primary importance, since it makes for a better mapping between items in the two domains. Although the details of this process vary among the different phenomena, this paradigm serves to unify a number of different results.

For example, suggestiveness and memorability may both be interpreted in terms of a high degree of similarity between the features of a name and its referent, with high distinctiveness among names and referents reducing the possibilities of confusion on either end. And the analysis easily extends to include non-words, since those without semantics map their *surface* features onto the semantic features of their referents.

The role of syntactic and pragmatic features is analogous, but the issue there is not simply one of how similar the two sets of features are, but also of how, for example, the selection restrictions of a name mesh with the rules of the command language. Where the analysis will lead in those domains is a question I am currently pursuing.

4. Conclusion

Thus it can be seen that, while syntax and discourse structure are important phenomena in human-computer communication, the lexical semantics of command languages is of equal importance and interest. The names which users or system designers give to the actions and objects in a command language can greatly faciliate or impair a system's usefulness. Furthermore, similar issues of semantic relations, deixis, ambiguity, etc. occur with the lexical items of command languages as in natural language. This suggests both that linguistic theory may be of practical aid to system designers, and that the complex lexical phenomena of command languages may be of interest to linguists.

References

Appelt, D. 1983. Planning English referring expressions. Technical Note 312.SRI International.Merlo Park, CA.

Barnard, P., N. Hammond, J. Morton, and J. Long. 1981. Consistency and compatibility in human-computer dialogue. *Int. J. of Man-Machine Studies*. 15:87-134.

Barnard, P., N. Hammond, A. MacLean, and J. Morton. 1982 Learning and remembering interactivecommands in a text-editing task. *Behaviour and Information Technology*. 1:347-358.

Benbasat, I., and Y. Wand. 1984. Command abbreviation behavior in human-computer interaction. *Comm. ACM*. 27(4): 376-383.

Black, J., and T. Moran. 1982. Learning and remembering command names. *Proc. Conference on Human Factors in Computing Systems*. (Gaithersburg, Maryland). pp. 8-11.

Bolinger D. 1982. The atomization of meaning. *Language*. 41:555-573.

Carroll, J. 1982. Learning, using, and designing filenames and command paradigms. *Behaviour and Information Technology*. 1:327-348.

Dumais, S., and T. Landauer. 1982. Psychological investigations of natural terminology for command and query languages. in A. Badre and B. Shneiderman, eds., *Directions in Human/Computer Interaction*. Norwood, NJ: Ablex.

Dumais, S., and T. Landauer. 1983. Using examples to describe categories. *Proc. CHI'83 Conference on Human Factors in Computing Systems*. (Boston). pp. 112-115.

Durham, I., D. Lamb, and J. Saxe. 1983. Spelling correction in user interfaces. *Comm. ACM*. 26(10): 764-773.

Finin, T. 1982. The interpretation of nominal compounds in discourse. Technical Report MS-CIS-82-03. University of Pennsylvania Dept. of Computer and Information Science, Philadelphia, PA.

Furnas, G., T. Landauer, L.Gomez, and S. Dumais. 1983. Statistical semantics: analysis of the potential performance of key-word information systems. *Bell System Technical Journal*. 62(6):1753-1806.

Hemenway, K. 1982. Psychological issues in the use of icons in command menus. *Proc. Conference on Human Factors in Computing Systems*. (Gaithersburg, Maryland). pp. 20-24.

Landauer, T., K. Galotti, and S. Hartwell. 1983. Natural command names and initial learning: a study of text-editing terms. *Comm. ACM*. 26(7): 495-503.

Rosenberg, J. 1982. Evaluating the suggestiveness of command names. *Behaviour and Information Technology*.1:371-400.

Rosenberg, J. 1983. A featural approach to command names. *Proc. CHI'83 Conference on Human Factors in Computing Systems*. (Boston). pp. 116-119.

Rosenberg, J., and T. Moran. 1984. Generic commands. *Proc. First IFIP Conference on Human-Computer Interaction*. London, September 1984.

Scapin, D. 1981. Computer commands in restricted natural language: some aspects of memory and experience. *Human Factors*. 23:365-375.

Scapin, D. 1982. Generation effect, structuring and computer commands. *Behaviour and Information Technology*. 1:401-410.

Tversky, A. 1977. Features of similarity. *Psychological Review*. 84:327-352.

Tversky, A. 1979. Studies in similarity. In E. Rosch and B. Lloyd, eds., *Cognition and Categorization*. Hillsdale, NJ: Erlbaum.