User Modelling and Natural Language Interface Design

Jaime G. Carbonell
Carnegie-Mellon University

Abstract

The design of friendly, robust man-machine interfaces is of paramount importance as computers become widely available to non-expert users. This paper examines the role of modeling the capabilities and knowledge of the expected user population as an important step in the design process. The development of robust natural language interfaces is advocated for casual or inexperienced users. Design criteria for effective natural language communication with existing software products and expert systems are proposed.

1. Introduction

As the widespread availability of computers for office, home, school and industry increases, so does the need for human-oriented, friendly interfaces. Whereas not too long ago professional programmers formed the bulk of the computer user community, more recently large numbers of office workers, managers, students in diverse fields, and other "casual users" are becoming the numerically-dominant segment of the user population. The micro-computer revolution has made computing hardware universally *available*; now a similar revolution in software is needed to make the computer universally *usable*. A significant part of the required software advances must take place in the area of friendly, robust man-machine interfaces.

Many a user has had to contend with the intricacies and idiosyncrasies of operating system interfaces, or the nightmare of a job-control language. Mostly, one just mimicked the incantations of more experienced users, and, after many readings of ambiguous manuals and much trial and error, one gradually became an expert user. Then, if a new system came on line, the entire familiarization process had to be repeated. This painstaking process was accepted as necessary by programmers and frequent users of packaged systems; it was a requisite investment whose dividends were measured in terms of services provided by the computer. However, many professionals in fields not requiring extensive use of the computer concluded that the familiarization cost was greater than any potential benefits, and hence chose not to use the computer. Thus, a primary factor restricting the utility and widespread acceptance of computers is the difficulty that non-expert users encounter in mastering the intricacies of unnatural man-machine interfaces. Clearly, it is imperative to design and implement interfaces to existing software systems that do not require excessive training or familiarity in relation to the expected frequency of usage by a particular user population. Equally clear is the fact that diverse users place different demands upon the interface, have different knowledge of the underlying system, and have different modes of interaction that enhance their comfort or productivity.

2. User Modelling

In order to ascertain the needs and abilities of a user vis a vis a particular interface design, it often proves useful to model the user in a manner that predicts how well a particular interface design will function. But, exactly what does a *user model* entail? There are two general classes of user models:

2.1. Empirical Quantitative Models

The empirical models are abstract formalizations of a general class of users defined in terms of the the design parameters of a user interface. For instance the KEYSTROKE model of Card, Moran and Newell [6] falls under this category. Loosely stated, that model asserts that the number of keystrokes is a determinant criterion of how productive a user can be at a not-very-demanding task, such as searching for information or performing a well-defined text editing procedure. Of course, this characterization is an oversimplification of the quantitative modeling technique, and the reader is referred to [6] for a comprehensive discussion of an entire class of models of this genre.

The quantitative models are based on empirical data compiled over many sessions with users and encode ***quantitative*** relations between the individual primitive steps a user takes to perform a potentially large task (given a fixed interface design) and the total time or accuracy of his performance. Such models are used to improve upon a proposed design by predicting the time required to perform a task, or its expected error rate with a given set of design parameters. Then, these parameters are changed in a direction indicated by the model to minimize time or performance errors. For instance, the ZOG system [11], a menu-selection frame-based interface for rapid data access and update was subjected to this type of analysis.

To summarize, the empirical predictive models do not attempt to simulate the internal reasoning or knowledge of the user, but rather correlate his or her external performance with given design parameters in a quantitative fashion. In contrast, the analytical models discussed below, attempt to simulate aspects of the users cognitive behavior in a more qualitative manner.

2.2. Analytical Cognitive Models

A major concern in Cognitive Science research is the simulation of human thought processes. Therefore, it is only natural that simulation techniques be applied to the task of understanding some of the user's relevant cognitive processes in order to better design and build effective interfaces. Rather than measuring external quantifiable variables (such as number of keystrokes, errors, or reaction times), cognitive modeling starts with a formal representation of the user's knowledge of the underlying task. For instance, in designing a data-base query interface, the cognitive modeller asks first and foremost the following types of questions:

- "How does the user's understanding of the information stored in the data base differ from the manner in which it is encoded?"
- "Is the user aware of the types of information encoded? (facts?, relations?, processes?)"
- "Does the user know what operations are available to extract the information (e.g., relational algebra operators)?"

In order to answer these questions, one must study the user more closely. First, it is unlikely that anyone's internal knowledge representation parallels the encoding of relational data bases.

Thus, part of the user's task is to perform this encoding conversion when data is extracted. And, a useful measure consists of examining how easily and how accurately users are able to perform the encoding transformation. Second, there are users who may need to *browse* through the data base in order to ascertain the type of information encoded, whereas other users may well be more knowledgeable or directed and may wish only to access specific facts. Third, it has been found that it is extremely difficult for untrained users to become experts at generating formal queries, but it is quite simple for them to state their query in natural language. Therefore, in this example, the user interface must facilitate the translation of information from natural language to formal query and from formal relational data base encoding back into natural language. Furthermore, a mechanism must be provided for enabling a user to query the categories of information available in a particular data base -- a mechanism analogous to a table of contents in a book.

In addition, a general analytical user model must address the following issues critical to the interface design:

- *The familiarity of the user with the functionality of the underlying system* -- An new interface to an operating system such as UNIX would be designed differently if it must accommodate users who know nothing of operating systems in general (in fact constructing such an interface would be an extremely difficult task requiring that the interface instruct the user on basic principles of interacting with operating systems).
- *The long-term commitment of the user* -- If the user is likely to use the system over a long period of time, it may prove cost-effective for him to learn a precise, terse interaction language. Whereas, if he is only an occasional or one-time user, an interface more akin to natural language is in order, requiring no training.
- *The range of sophistication among the user population* -- If experts and novices must share a common interface, it should be designed to hide much of the complexity from the novice but provide all the functionality required by the expert. Moreover, the interface capabilities should grow gradually as the novice gains experience and becomes an expert.
- *The user as an interface designer* -- If the user is likely to be experienced, the system should provide a facility for personalizing and extending the interface. In natural language interfaces this may take the form of allowing the user to input new dictionary entries or grammatical structures. In a text editor such as EMACS, it can take the form of allowing the user to define his own key-board macros and additional commands. However, such complexity is precisely the type of information that must remain hidden from the novice user. Furthermore, the system should *not* encourage divergent user personalization in the function performed by the underlying system, lest the actions of multiple users become incompatible at a more fundamental level.
- *Mixed-initiative capabilities* -- A well-designed interface should enable the user to ask for help ay any point in his interaction. The form of this help depends on the sophistication of the user. An expert may only wish to be reminded of his options at a given point, and would be annoyed and delayed by a lengthy explanation, whereas a novice may require examples of the basic options (and may be confused by a lengthy enumeration and explanation of all possibie options). Hence, in providing help, in formulating responses and in presenting more of the complexities of the system, the interface must have some knowledge of the level of expertise of the user.

The cognitive approach to modeling is the first step in deciding the overall structure of an interface, whereas the quantitative approach attempts to optimize parameters within a certain chosen design. Hence, both types of user modeling are necessary in designing convivial user interfaces. Now, let us turn to additional design criteria relevant to user-friendly interfaces in general, and natural language interfaces in particular.

3. User-Friendly Interfaces

The type of user-friendly interface we envision is targeted at the user who is either computer-naive or at best unfamiliar with the particular system he needs to use. However, an understanding of the underlying task the user sets out to perform is assumed. Thus, a statistician or an economist who interacts with a statistical analysis package should know what operations to perform on his data, but need not worry about subroutine calling sequences, details of the file management subsystem, argument passing conventions, etc. Similarly, a secretary interacting with an operating system interface should know its general function, but need not learn an artificial stylized command language. For instance, the following dialog exemplifies a user-friendly, natural language interface: (User input appears in *italics*, and the translation of the natural language command into the system's formal command language appears in [square brackets].)

```
What fortran files do I have?
[Dir *.for<cr>]
        GAUSS.FOR
        GAUSS2.FOR
        MATRIX.FOR

>Transfer GAUSS.FOR from my directory to the system library.
[Copy GAUSS.FOR <SUBSYS>GAUSS.FOR<cr>]
[Delete GAUSS.FOR<cr>]
        ERROR: DIRECTORY ACCESS PRIVILEGES REQUIRED
        'Copy' and 'Delete' commands not performed.

>print that file on the Diablo
'That file' = GAUSS.FOR? (Yes or No): yes
[Diablo<cr>]
[*GAUSS.FOR/print/notify<cr>]
        <SMITH>GAUSS.FOR queued

>Has my file been printed yet?
[Info diablo<cr>]
        <JONES>LETTER.JAN
        <SMITH>GAUSS.FOR
No, GAUSS.FOR is still on the Diablo queue.

>Cancel it.
'It' = Your command to print GAUSS.FOR on the Diablo?:   yes
[Diablo<cr>]
[*GAUSS.FOR/cancel<cr>]
        <SMITH>GAUSS.FOR canceled
```

The dialog above illustrates a cooperative natural-language interface that we have built with present natural language processing technology (Using a variant of the DYPAR-II parser [5, 9]). The interface system can echo the command or command sequence that is executed as a result of interpreting the natural language utterance. Thus, a novice user can be trained on the use of a formal, more terse command language all the while he is performing useful tasks by communicating in a subset of natural language. Additionally, one should note that natural language allows for referential ambiguity and a system must be capable of handling such input in a graceful manner. In the example illustrated above, the system asked for confirmation of its referential hypotheses when there was room for ambiguity.

The interface exemplified above is indicative of the type of robust, task-oriented natural language processing that can be developed with present-day technology. In addition to the DYPAR-II project at Carnegie-Mellon University, we have two other experimental systems targeted at developing similar interfaces. The COUSIN project (COoperative USer INterface) provides a uniform interface with the UNIX operating system, as well as the SPICE (Scientific Personal Integrated Computing Environment) single-user computers. It is close to becoming operational, but it exhibits rather limited natural-language communication. The MULTIPAR project [4, 8] addresses the problem of robust task-oriented natural language interfaces. The rest of our discussion centers on the development of robust natural language communication as an extremely useful tool for implementing flexible man-machine interfaces.

4. Criteria for a Friendly Natural Language Interface

In order for a natural language interface to be usable and friendly to non-expert users, as well exhibit a degree of large-scale applicability, it should meet the following criteria:

- *Syntactic coverage* -- A parser not capable of syntactic segmentation of simple English utterances is of little use in any application. Fortunately, computational aspects of syntactic parsing are well understood. Moreover, the syntactic structure of commands, queries and assertions that typify user interactions with a natural language interface is invariably simple. Experience has shown that the much more complex syntax present in textual or other written material is simply not found in task-oriented man-machine dialogs.
- *Task-oriented semantic coverage* -- Each application domain for natural language interfaces exhibits a well-defined semantic model. For instance, the commands given to an operating system, or the set of permissible updates to a data-base file manager form a bounded set of well understood operations. The natural language interface can exploit these restrictive domain semantics creating a much more effective and robust system than one can hope to create for unrestricted English.
- *Flexibility in the presence of extra-grammaticality* -- Experiments show that almost half of all utterances typed at a natural language interface are grammatically deficient. Problems range from misspelled words to ellipsed sentence fragments, missing punctuation, interjections, and transposed words or phrases. The vast majority of these problems present no problem to a human interpreter, and therefore ought to be tractable by an automated analyzer.
- *Semantic resilience* -- Knowledge of the underlying domain should be exploited to resolve ambiguity. For instance the utterance "cancel it" in the preceding dialog is many-ways ambiguous. In general, almost anything can be cancelled. In the operating-system domain, only a restricted class of commands can be cancelled. Moreover, in the context of the preceding utterances, it only makes sense to cancel the queued printing request. Making this deduction requires knowledge of the domain as well as the structure of a dialog.
- *User friendliness* -- Fulfilling the dual goals of providing maximal assistance to a naive user, and being unobstrusive to a more experienced user is a difficult challenge. One possibility is to enable the user to mix natural language commands with the terser system commands as he learns them. Another avenue to resolve this issue is to present the user with default options when ambiguity or other troubles arise -- saving the user from retyping the entire

input while retaining control over all actions generated by the interface.

- *Transportability* -- An interface must be applicable to many different domains, in order to justify the software development costs, and to provide uniform access to multiple software facilities. This objective clashes with the semantic resilience goal, which requires that the interface have access to an underlying semantic domain model. Here, we advocate the representation of the semantic model as a data structure read in by the interface along with the dictionary at load time. Thus, the parser, syntactic knowledge, dialog-structure knowledge, and domain semantics are separate modules -- the latter being an interchangeable data file. An interface need only understand commands to a given subsystem at one time (plus an ability to switch subsystem and interface), therefore no problems of unforeseen interactions across domains need arise.

5. Towards Robust, Multi-Strategy Parsing

Current work in computational linguistics indicates that it possible to design a friendly natural language interface in accordance with the criteria listed above. The approach we have taken is primarily a synthesis of previous natural language parsing techniques, together with a set of "fail-soft" recovery heuristics. Whereas syntactic parsing methods (e.g. ATNs [13]) capture linguistic regularities in a general manner, semantic grammars [10, 2] encode domain-specific semantics, and expectation-based parsing [12, 1] is quite useful for general semantic disambiguation, no single technique is capable of handling all aspects of natural language analysis. Moreover, experience has shown that the strengths of one technique overlap with the shortcomings of other parsing techniques. Therefore, taking a pragmatic approach, we have chosen to synthesize the best aspects of each technique into an integrated multi-strategy parser -- MULTIPAR [9].

Although, MULTIPAR is still under development, its predecessors have taught us some useful lessons -- and have themselves served as flexible natural language interfaces. Past systems developed at CMU include:

1. FLEXP -- A recursive pattern-matching flexible parser applied to an advanced message system [7] demonstrated the need for combining bottom-up and top-down parsing strategies when faced with input that deviates from a prescribed grammar.
2. CASPAR -- A case-frame parser demonstrated the power of domain semantics in parsing both correct and extragrammatical input [9]. Moreover, selective treatment of constituents on the basis of ease of recognition has proven a great help in realigning a parse when an incomprehensible segment is encountered.
3. DYPAR -- A three-strategy parser demonstrated the feasibility of combining more than one parsing strategy into a unified system. Strategy selection occurs on the basis of the expected form of embedded constituents. DYPAR is currently in use as natural language interface to a simple semantic-network data base access and update, a factory scheduling and simulation expert system, and a light-bulb manufacturing process data base and simulation system. The field-testing of DYPAR has indicated a need for a more flexible interface, one in which sophisticated ellipsis and anaphora resolution - as well as additional focused recovery methods can be implemented. More recently, DYPAR-II has been extended to serve as a natural language interface to the XSEL/R1 expert system at Digital Equipment Corporation [5] This experience is instrumental in the current design of MULTIPAR.

The type of natural language phenomena that one must handle in a robust flexible interface include the following set:

- *Spelling correction* -- About 40% of all user errors are careless misspellings. Context-free spelling correctors can handle a large number of cases; however, more sophisticated methods are required for human-like performance. For instance, the following sentence was encountered in one of our studies: ***"Transfer the fortran flies to the accounts directory."*** It is obvious that the person meant to type "files", but accidentally transposed letters. However, "flies" is a correctly spelled word -- one present in the task-domain which happened to be statistics on med-fly infestation. No self-respecting spelling correction algorithm would then try to correct a correctly spelled word appropriate in the general context. Clearly, we need *context-sensitive* spelling correction. The word "flies" is semantically inappropriate in its specific location in the sentence. We must have a spelling correction method capable of making this type of judgment.
- *Definite noun phrases and anaphora* -- The sample dialog in the previous section illustrated both of these phenomena. When a user types "that file", or "my file", the system must resolve the referent to a specific entity in the preceding dialog. The same problem occurs (without explicit type restrictions) when the user types "it" or "that" as an anaphoric referent. It is imperative for a user-friendly interface to resolve these referents. Merely complaining to the user that he is not being specific enough is a non-operational solution. We attempted such a solution, but only succeeded in creating frustrated and irate users. For example, a user typed ***"Do I have a FOO.REL file?"*** And, after an affirmative response, he typed ***"OK, now load it into core."*** The system complained that "it" could not be resolved, whereupon after a long pause, the user typed ***"Do a load."*** Clearly the latter sentence is more terse, and harder to interpret (Load what? Load it where?). It was obvious to the user what file he was referring to; therefore it ought to have been obvious to the system as well.
- *Ellipsis and fragmentary input* -- It is often the case that people utter sentence fragments which make sense only in the context of an on-going dialog. The same phenomenon occurs in natural language interfaces. The vast majority of these fragments can be resolved by reinstantiating the preceding utterance with the new fragment substituting for the semantically corresponding constituent of the previous sentence. For instance, a typical ellipsis occurs as follows:

  Copy the fortran files to my directory.
  Now the data files.

  It is clear that "data files" substitutes for "fortran files" and a reexecution of the previous command is called for. A case frame strategy with semantic constraints can handle this type of ellipsis easily. However, more difficult forms of ellipsed commands occur, such as:

  Copy the fortran files to my directory.
  I meant to the accounts directory.

  Here, the system must understand how to undo its previous command (e.g., deleting the object case from the destination case of a "copy" undoes the effects of a copy") as well as performing the prior ellipsis resolution.
- *Fail-soft recovery Heuristics* -- When a fragment of a user's utterance is unparsable, the interface system should not abandon hope. We are developing heuristics for bridging difficult segments, realigning the parse in a bottom-up manner, and returning to the troublesome part later with additional syntactic and semantic constraints from the rest of the sentence. For instance, case-selection restrictions can be relaxed if other cases have been correctly instantiated, and pattern matches can be made partial matches, as long as

the set of possible constituents that could match has been reduced to a small set by the surrounding context [3].

- *Focused interaction* -- When recovery heuristics or semantics-based disambiguation fail, a robust interface must interact with the user to produce appropriate clarifications. These interactions should be focused on the source of the problem, and the system should present alternatives or default choices in a concise manner.

In the MULTIPAR project we are striving to achieve these objectives. The fact that we are developing a flexible control structure to enable different parsing strategies to be brought to bear at appropriate times facilitates the introduction of ellipsis resolution, anaphora resolution, and fail-soft recovery mechanisms.

6. Concluding Remark

In order for a natural language interface to be truly useful and accepted by a wide user community it must be robust and tolerant of user errors -- especially when the user does not consider terse grammar and fragmentary input to be in error. Rather, users typically wish to type as little as necessary to get their message across. Therefore, natural language interfaces must accept anaphora, ellipsis and other means of abbreviating utterances. Moreover, a natural language interface that is incapable of understanding simple, if ungrammatical utterances causes a naive user do distrust the system as a whole. Hence our argument in favor of natural language interfaces must be augmented with the proviso that the interfaces be substantially well designed and robust enough to gain general acceptance. As our present work indicates, the basic technology required to build flexible interfaces in semantically well-defined domains exists, but must be refined and developed into working systems. The design constraints on the natural language interfaces were developed by examining and modeling the capabilities and performance of users, especially the novice users that constitute a rapidly expanding segment of the user population.

7. References

1. Birnbaum, L. and Selfridge, M. Conceptual Analysis in Natural Language. In *Inside Computer Understanding*, R. Schank and C. Riesbeck, Eds., New Jersey: Erlbaum Assoc., 1980, pp. 318-353.

2. Burton, R. R. Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems. Tech. Rept. 3453, Bolt Beranek and Newman, 1975.

3. Carbonell, J. G. Towards a Self-Extending Parser. Proceedings of the 17th Meeting of the Association for Computational Linguistics, ACL-79, 1979, pp. 3-7.

4. Carbonell, J. G. and Hayes, P. J. Dynamic Strategy Selection in Flexible Parsing. ACL81proc, ACL81, 1981.

5. Carbonell, J. G., Boggs, W. M., Mauldin, M. L. and Anick, P. G. The XCALIBUR Project, A Natural Language Interface to Expert Systems. Proceedings of the Eighth International Joint Conference on Artificial Intelligence, 1983. (Submitted)

6. Card, S. K., Moran, T. P. and Newell, A.. *The Psychology of Human-Computer Interaction.* Erlbaum Assoc., Hillsdale, NJ, 1983.

7. Hayes, P. J. and Mouradian, G. V. Flexible Parsing. Proceedings of the 18th Meeting of the Association for Computational Linguistics, ACL80, 1980, pp. 97-103.

8. Hayes, P. J., and Carbonell, J. G. Multi-Strategy Construction-Specific Parsing for Flexible Data Base Query and Update. IJCAIVIIproc, IJCAI-81, August, 1981, pp. 432-439.

9. Hayes, P. J. and Carbonell, J. G. Multi-Strategy Parsing and it Role in Robust Man-Machine Communication. Tech. Rept. CMU-CS-81-118, Carnegie-Mellon University Computer Science Department, May, 1981.

10. Hendrix, G. G., Sacerdoti, E. D. and Slocum, J. Developing a Natural Language Interface to Complex Data. Tech. Rept. Artificial Intelligence Center., SRI International, 1976.

11. Newell, A., McCracken, D. L., and Akscyn, R. M. ZOG and the USS Carl Vinson. In *Computer Science Research Review*, Carnegie-Mellon University, 1980-1981.

12. Riesbeck, C. and Schank, R. C. Comprehension by Computer: Expectation-Based Analysis of Sentences in Context. Tech. Rept. 78, Computer Science Department, Yale University, 1976.

13. Woods, W., Kaplan, R. and Nash-Webber, B. The Lunar Sciences Natural Language Information System: Final Report. Tech. Rept. 2378, Bolt Beranek and Newman Report, 1972.

Address: Prof. Jaime G. Carbonell
Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213
U. S. A.