Intelligent Man/Machine Interfaces on the Space Station

Rodney S. Daughtrey
Research Associate, Johnson Research Center
Research Institute, Room A-11
University of Alabama in Huntsville
Huntsville, AL 35899
(205)-895-6217

ABSTRACT

The computer systems that will be resident on the space station will necessarily be large, complex configurations of software and/or hardware which must function in concert to maintain the operations of the mission. However, much of the complexity should be transparent to the user of those systems, allowing the user to concentrate as much as possible on the purpose of his/her interaction with the computer, rather than unnecessary detail.

This paper addresses some important topics in the development of good, intelligent, usable man/machine interfaces for the space station. These computer interfaces should adhere strictly to three concepts or doctrines: generality, simplicity, and elegance (just as the programming language code below these interfaces should follow). Generality refers to the commonality of usage and the similarity of form and function that should predominate all the interfaces, so that the user is provided with computer working environments that appear and perform in much the same way. Simplicity is obviously desirable, but not a concept to be taken lightly. It is very important that the interfaces simplify operations wherever possible (and desirable), and make intelligent inferences about the intent of the user to save time and unnecessary attention to detail. Finally, the elegance of the interfaces should be of great concern. The interfaces should be as concise as possible, exhibiting the "principle of least astonishment".

The author will also discuss the motivation for natural language interfaces and their use and value on the space station, both now and in the future.

AI provides an extremely powerful tool with which to think about and develop software applications to run on the space station. But, without well-thought-out, intelligent, truly usable man/machine interfaces to harness this asset, much of this power will be lost.

INTRODUCTION: LEVELS OF COMMUNICATION AMONG HUMANS AND COMPUTERS

Communication is an interesting subject. Whether we communicate with other humans, machines, or some other entity, we do so to achieve some end. With computers, we attempt to get them to do things for us, (usually) saving time and effort on our part. It only makes sense, then, that getting them to do something should be made to be as effortless and logical as possible. How should we design them in order to maximize ease of use? Should we make them like us?

Figure 1 shows a comparison between human levels of communication and machine levels of communication (as used by humans).

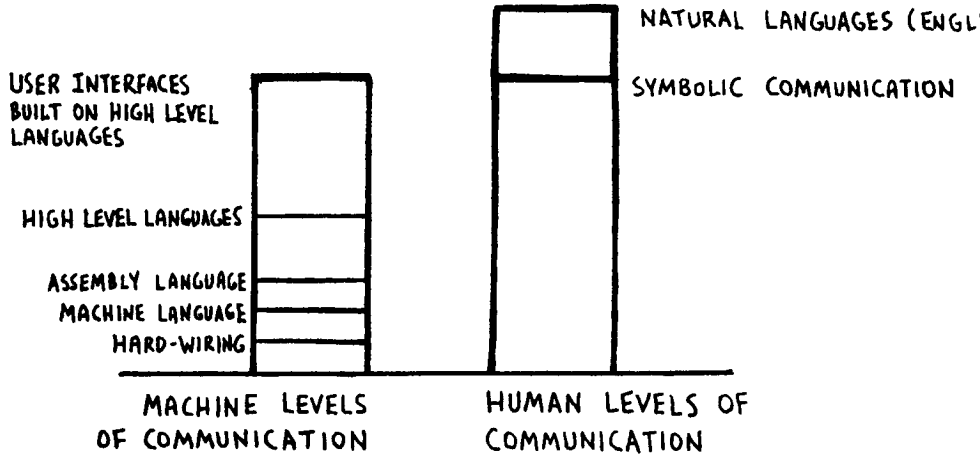

Fig. 1. Comparison of human and machine communication levels

At the bottom of machine communication we have hard-wiring, or direct communication with the physical parts making up the computer. This was done with the ENIAC computer in the late 1940's. The important thing to realize here is that there is no level of abstraction of input, and therefore no user interface, between the human and machine at this level: the human operator is responsible for explicitly determining the actual flow of electricity through the machine. This may seem rather silly today, but it was at one time the price one had to pay for "automated" computing.

Machine language soon followed hard-wiring and abstracted most of the physical aspects of programming a computer, thereby becoming the first user interface. Then came assembly language, which gave mnemonic names to machine language instructions and provided a slightly more forgiving format for writing programs. High-level languages such as FORTRAN and LISP appeared on the scene next, making programming a much simpler and less tedious task and allowing programmers to be more productive and to write programs that were less machine dependent than those in the past.

Much of the software written today makes use of the more friendly and intuitive input methods that have been developed for today's computers (the mouse, user menus, icons, multiple windows on the screen, etc.). This software often reaches a level of communication that humans use in their everyday life, a level the author calls "human symbolic communication". Programs are beginning to interact with the user in the same way that a human interacts with and responds to symbolic stimuli in the real world (traffic lights, road signs, a telephone ringing), making the programs easier to learn and use.

Finally, interfaces which communicate with the user in plain English are currently under a great deal of study and development. Some progress has been made, but many large, difficult problems still remain. The importance and relevance of natural language interfaces to the space station will be discussed later in this paper.

ESSENTIAL ELEMENTS OF A USER INTERFACE

Regardless of the type of interface (be it graphical, natural language, or some other), three principles (in the author's view) should predominate its form and function: generality, simplicity, and elegance [2].

What was the author's motivation behind concentrating on these three ideals? It stems from the fact that these ideals are (usually) the three main goals to strive for when writing computer programs. A programmer's ultimate objective should be to write programs that are easy to understand, easy to modify, and easy to use. Since these principles are also what a good user interface should embody, the same philosophies should apply.

The principle of "generality" might better be termed the principle of "non-specificity". All the user interfaces on the space station (or as many as is feasible) should look, be used, and perform in much the same way, even though they might perform vastly different tasks. These interfaces should each be as non-specific as possible, so that features, commands, and utilities resident in one interface will most likely appear in all the interfaces. This cannot always be the case, of course; some features and commands in one interface may not even make sense in another context. If possible, the similarity of features and commands should extend both functionally and graphically across interfaces; in other words, not only should features common to more than one interface perform the same way, they should even appear in the same form on the screen. The main idea behind generality, of course, is that once a user becomes familiar with one interface, he/she would be able to learn to use other interfaces in much less time and with considerably less effort.

Incorporating simplicity into a user interface may seem

rather an obvious objective, but a fine line lies between keeping things simple and lessening the functional power of the interface. Einstein may have said it best: "Everyhing should be made as simple as possible, but not simpler" [1]. At any rate, a good rule to follow might be that wherever unnecessary detail can be suppressed, it should be, but not at the functional expense of the user. A simple example involves the user input of several parameters for some computing task (a statistical program, say). If the user needs to run the program more than once, he/she definitely should have the program option of supplying the same parameters as on the first execution. Not having this option would violate the concept of simplicity; either the user would be subjected to unnecessary detail (i.e. the program forces the user to input all the parameters again), or the user would lose functional power (the program only allows the same values to be used again on subsequent executions).

Implementing elegance into an interface is a highly subjective task. Elegance is really a combination of good taste and common sense, and although it may mean different things to different programmers, a few guidelines do exist.

Probably the main practice that should be followed is the adherence to the "principle of least astonishment". For those unfamiliar with this principle, it states that the programmer should devote considerable attention to the naming of commands, features, program options, etc. in order to make as obvious as possible what operation or concept is meant by that name. Put simply, "say what you mean". The idea is that the user should be "least astonished" at what the command, feature, or program option implies. Although this is stated tongue-in-cheek, anyone who has worked with software in which the keystroke sequence "CTRL-J CTRL-M ESCAPE R47" was needed to save a file, rather than, say, "CTRL-S", can appreciate its importance.

WHAT SHOULD USER INTERFACES ON THE SPACE STATION BE LIKE?

Are these graphical, symbolic user interfaces discussed earlier the way to approach user interfaces on the space station? Can we develop even higher level natural language interfaces which can communicate with us in English and truly understand our instructions and intentions? Assuming we can, when and under what conditions should we develop them? To answer these questions, we need to identify the motivations for both types of interfaces and the advantages and disadvantages of each.

With graphic-oriented interfaces, speed of use and conciseness of expression are definite advantages over natural language interfaces. Users can accomplish tasks much faster using input devices like the mouse, menus, windows, and other similar features. Tasks that would have to be specified in sentence form in a natural language interface could be effected

with a single mouse click or some keystroke.

Natural language interfaces, however, are not without their advantages. With graphic-oriented interfaces, it is sometimes very difficult or impossible to perform some task that was unforeseen at the time of the building of that interface. Given a powerful enough natural language interface, the user is given the power of completeness of expression: he/she can specify any task needed to be performed through English text (again, assuming that such an interface is implementable). Another argument for natural language interfaces is that there is virtually nothing to learn about the interface for the user (assuming he/she knows English).

Considering the above, it seems logical to make the following conclusions:

1) Both graphic-oriented and natural language interfaces should reside on the space station (assuming powerful enough natural language interfaces can be developed).

2) Graphic-oriented interfaces should be used in conjunction with tasks that are well understood and bounded in terms of previously foreseen needs and capabilities.

3) Tasks characterized by complexity and possible unknown but essential requirements should have both graphic-oriented and natural language interfaces (again assuming their existence). The graphic-oriented interface would be the interface normally used, with the natural language interface used for any appropriate situation.

4) As the space station program develops further, more natural language extensions to the existing graphic-oriented interfaces should be developed to accomodate the greater variety of people that will be participating in the program.

Without a doubt, the space station will contain an amazing amount of computational power, and harnessing that power and making it usable should be a huge consideration. The complexity of the entire operation and the fact that a large variety of people must work together in the same environment should demand a great amount of forethought about the man/machine interface.

REFERENCES

1. Minsky, M. The Society of Mind. Simon and Schuster, New York, New York, 1985, p. 17.

2. Schneider, G.M, and Bruell, S.C. Advanced Programming and Problem Solving in Pascal. John Wiley and Sons, New York, New York, 1981, p. 52.