S7-61

N96- 12922

p- 9

CLIPS: A Proposal for Improved Usability

Charles R. Patton
Computer Sciences Corporation
16511 Space Center Blvd.
Houston, TX 77058

This paper proposes the enhancement of the CLIPS user interface to improve the over-all usability of the CLIPS development environment. It suggests some directions for the long term growth of the user interface, and discusses some specific strengths and weaknesses of the current CLIPS PC user interface.

Every user of CLIPS shares a common experience: their first interaction with the with the system itself. As with any new language, between the process of installing CLIPS on the appropriate computer and the completion of a large application, an intensive learning process takes place. For those with extensive programming knowledge and LISP backgrounds, this experience may have been mostly interesting and pleasant. Being familiar with products that are similar to CLIPS in many ways, these users enjoy a relatively short training period with the product. Already familiar with many of the functions they wish to employ, experienced users are free to focus on the capabilities of CLIPS that make it uniquely useful within their working environment.

To those without the benefits of such a background, however, the first meeting with CLIPS may have been more of a struggle than a triumph. Imagine the worst-case scenario for the aspiring CLIPS programmer. The inexperienced user may know little about rule based programming, so a fundamentally new programming style must be learned. The EMACS editor must be understood before any CLIPS code can be written. The nuances of the CLIPS language and its syntax must be mastered before the simplest program will compile. Testing a rule based system can be especially complex. A new operating system must be mastered. In short, the new CLIPS programmer must complete a lot of learning in a very short time.

Experience has taught us that modifications to the user interface of a software product can make that product both easier to learn and easier to use. A major goal of any changes to the CLIPS user interface would be to reduce the time required to learn the basics of the CLIPS development environment.

Additionally, enhancements to the CLIPS user interface could allow experienced programmers to develop software faster and more easily. Advances in user interface technology allow us to design interfaces specifically suited to multi-dimensional activities like developing rule-based software. Few managers would be opposed to improving the productivity of their programmers, provided the costs of the enhancements are not excessive relative to their benefits.

Another goal of this paper is to promote an awareness of usability issues among CLIPS users and developers. The purpose of these recommendations is to make the CLIPS community aware of some possible user interface enhancements for their development environment. The validity of the following usability recommendations will be established or refuted by CLIPS users. Certainly there are other ideas that will come directly from the users themselves, due to their extensive experience with the product. The user community can then discuss any possible enhancements with CLIPS developers, weighing matters such as costs, benefits, and priorities.

The CLIPS development group is constantly improving its product. As any product is made more powerful, however, it must also become more complex. Additional attention should be paid to the user interface of a product as its capabilities grow, because that product is making greater and greater demands upon the resources of its users. There is more to learn, more to do, and more to remember than there was before enhancements were made. For example, the object-oriented CLIPS system will be more complex than the current releases of this product. Enhancements to its user interface could reduce the amount of complexity presented to the user.

The development of the CLIPS window interface for the PC was a first step toward improving the usability of this product. The application of relatively new interface technologies such as the mouse pointing device and pull-down menus are distinct improvements over the basic command line interface. The window interface clearly saves typing time and reduces the cognitive load of the CLIPS user. While these steps are applauded, there are still aspects of the CLIPS user interface that demand improvement.

Proposed - A New CLIPS Development Environment

Certainly there can be no single CLIPS development environment. CLIPS is run on a variety of platforms in a number of different ways to solve a multitude of problems. Individuals have widely different programming styles that must be accommodated.

The idea behind this new development environment is to create a flexible user interface that can support the beginning user or be adapted to assist the experienced CLIPS programmer. Since the interface supports several different processes (editing, compiling, testing, etc.) a multi-window approach would be appropriate. Wherever possible, interface functions would be devised to reduce the cognitive load on the user.

Consider again the beginning CLIPS user. This person's primary activities are: writing simple programs, compiling them, and testing their functionality. A multi-window user interface can provide all of these capabilities at a glance, reducing the number of things that the user must remember how to access. (See Figure 1.) The user's CLIPS code would be available for editing in the window on the left. Interaction with the compiler and real-time testing would occur in the upper right window. A listing of the currently active facts (i.e. a "show facts" command) is displayed in the window on the lower right.

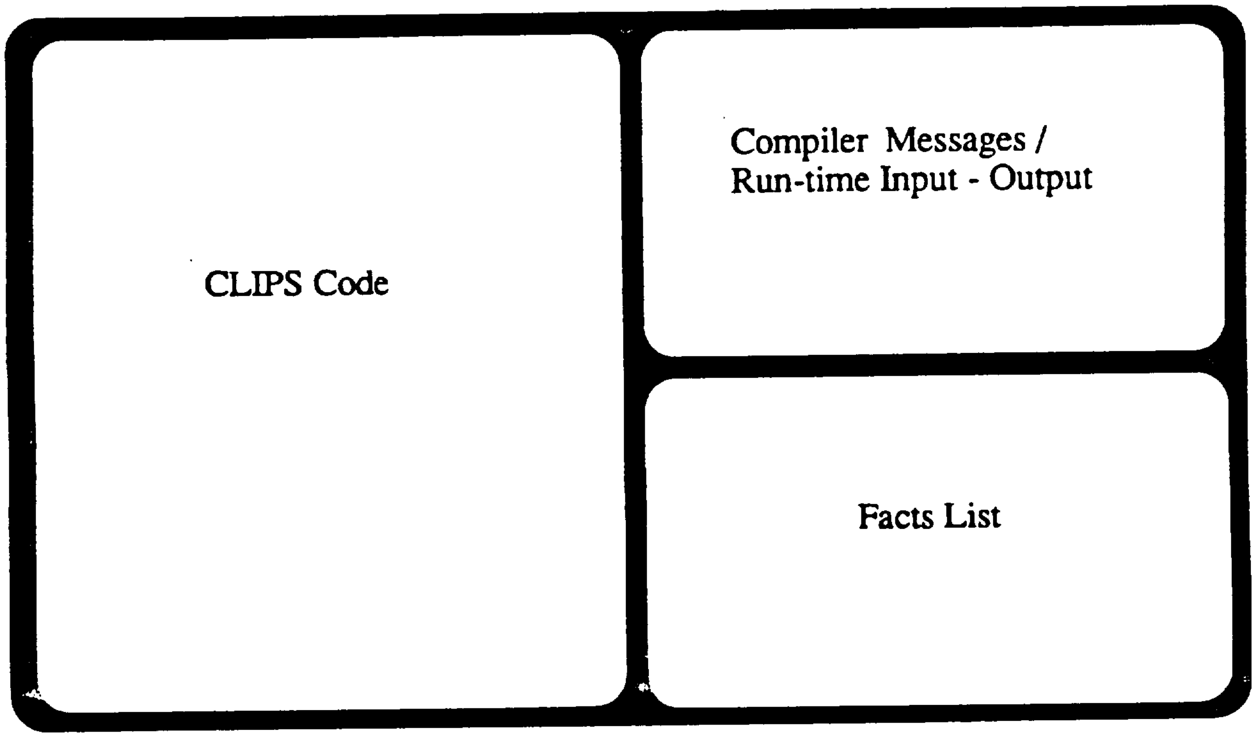

Figure 1: Example of a basic CLIPS development environment.

This display gives the programmer several interesting capabilities. It is possible to see and change the written code as it is compiled, reducing the time required to re-edit source code files. Program activities during testing can be traced back to the source code, speeding up the debugging process. The facts list would provide a constant display of the current facts that the system is using and generating. Here, then, most of the information that a beginning CLIPS programmer needs to know is available in one display. Less time is spent switching between modes and asking for information because it is all currently available on the screen. The user has fewer things to remember as the task is completed. The user can focus on the task at hand, rather than focusing on the processes involved in completing the task.

For CLIPS experts, the interface proposed in figure 1 would not be powerful enough to help them perform their tasks - in fact, it might even slow them down. Advanced users would require additional functionality, like the display shown in Figure 2. Notice that another window is available to display the source code from another program that references CLIPS rules as it runs. The facts file would support a initial list of facts to be used in testing a CLIPS module, while the current facts are again displayed in a facts list window.

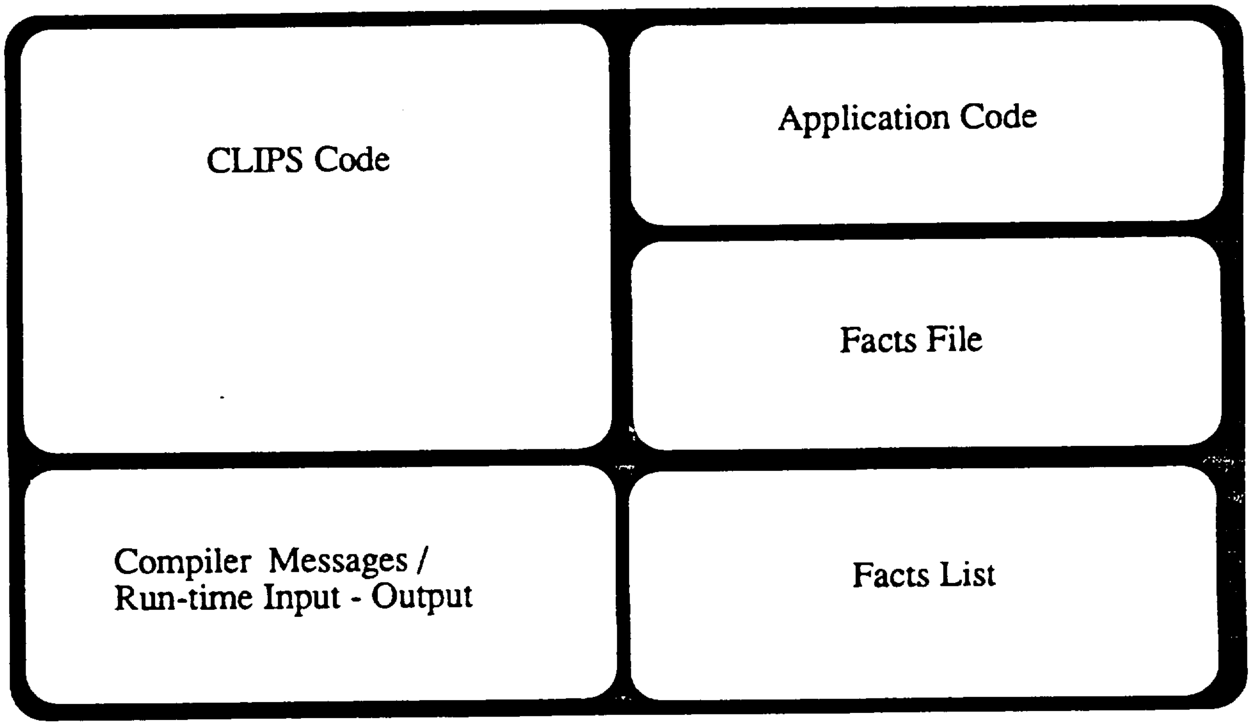

Figure 2: Example of an advanced CLIPS development environment.

Note: Please do not take figures 1 and 2 too literally. Window location and size would be under the user's control. The given arrangement is for the purposes of this discussion only.

Developing a generic user interface for CLIPS across its many platforms and operating systems would be technically challenging. Hardware constraints and portability requirements must certainly be considered. But as platforms become more powerful and as operating systems and as user interface management systems are standardized, ideas like this will become feasible.

Proposed - Changes to the CLIPS Development Environment

The following topics are presented as areas where the current CLIPS PC user interface might be improved upon. Specific recommendations and objective justification will be provided in further discussion of each issue.

Improving the format and content of compiler output.

Understanding compiler output is a critical aspect of learning a new computer language. No one really likes having their errors pointed out to them - especially by a machine. So it is important that compiler statements to the user be clear, accurate, and helpful.

```
a.   Compiling rule: grab-object-from-ladder
     Missing function declaration for defrule  <color highlight>

b.   Compiling rule:  drop-object-once-moved
     An argument in a function call must be a constant, variable, or expression

     ERROR:
     (defrule drop-object-once-moved " "
        ?f1 <- (goal-is-to-move  ?obj  ?place)
        ?f2 <- (monkey  ?place  ?on  ?obj)
        ?f3 <- (object  ?obj  ?  ?  light)
        =>
        (printout t "Monkey drops the  "  ?obj ."

c.   Compiling rule:   hold-object-to-move  +j +j +j +j
```

Figure 3: Examples of clear CLIPS compiler messages.

Figure 3 contains examples of some good CLIPS compiler messages. Notice in Example (a) that the system identifies the rule being compiled, and then follows the message with a statement of the problem. In the version of CLIPS used for this test, the error messages are printed in a separate color from the rest of the text. This is good for the on-line user with a color monitor, but notice that the effect is lost on the printout. The difference between the two types of statements could be further displayed by the use of italics or by flagging the error message with asterisks (**).

In Example (b), the compiler has printed the rule in question, up to the point of the error. This is a good practice, since it clarifies the position of the problem within the rule.

In Example (c), the +j symbols indicate that the rule has been compiled successfully. This allows the user concentrate on other rules that have syntactical problems.

```
1. Compiling rule: grab-object Function retract expected argument #1 to be of type
   number or variable

2. Compiling Region...
   Compiling rule: grab-object-from-ladder
                                            Expected ')' to finish rule or '(' to be

   gin new action
             Error:
                 defrule grab-object-from-ladder ""
                                                      ?f1 <-(goal-is-to han

    ds ?obj)
          ?f2 <- (object ?obj ?place ceiling light)...

3. Compiling rule:  unlock-chest-to-hold-object  +j +j +j

   Expected left parenthesis to begin defrule or deffacts statement
   Compiling rule: hold-chest-to-put-on floor  +j +j +j +j

   Found unrecognized construct...
```

Figure 4: Examples of unclear CLIPS compiler messages.

Figure 4 contains examples of compiler statements that are less clear, less readable, or potentially misleading. In Example 1, the rule name and the error message are not separated, making reading and interpreting the message more difficult.

Example 2 illustrates a very useful feature of the CLIPS compiler - the regional (incremental) compile. A specific section of a CLIPS program can be highlighted and compiled within the editor. This speeds up the compiling process, and allows users to complete and compile "one rule at a time". Notice, however, how difficult it is to distinguish between the error message and the display of the rule due to the awkward spacing of the statements. Ideally, this message would be formatted much like Example (b) in Figure 3.

Example 3 can be difficult for a novice CLIPS user to interpret. What the compiler is trying to say is that 2 rules: *unlock-chest-to-hold-object* and *hold-chest-to-put-on-floor* have compiled correctly, and that two rules (one after *unlock-chest* and one after *hold-chest*) have failed to compile. The rules are not named because they were never recognized as rules by the compiler. While there are some cues in the messages that rules were not compiled, they are not powerful ones. Redundant cues would assist the novice user without distracting the experienced user.

The CLIPS Editing Environment

Experienced programmers and computer users generally have their favorites among the wide variety of editors and word processors that are currently available. CLIPS currently allows the user to choose any standard text editor for preparing code, which permits an individual to select the preferred editing environment.

Many programmers are particularly fond of the EMACS editor, while others do not like it at all. For beginners in the CLIPS environment, EMACS is a poor choice since it requires the user to learn and remember a specific set of commands as they try to learn and remember CLIPS syntax. Doing both of these things at once is a particularly heavy load for the new CLIPS programmer. If a more modern, direct manipulation style editor were offered as an option for beginners, their training time could be reduced. Also, a custom CLIPS editor could have built-in functions that relate specifically to programming in CLIPS, significantly speeding up the typing / coding process. Specific examples of these custom functions will be discussed later.

A Command Storage Buffer and Function Key

One of the most common errors committed by CLIPS users occurs when a relatively long command is typed on the command line. If a typographical error occurs early in the command, and it is not detected immediately, the user is forced to delete the entire line and type the entire command over. This can be quite frustrating, particularly when a long command is in error only because the initial parenthesis is missing.

It would be feasible to store the contents typed on the command line in a buffer associated with a PF key. Essentially, this would permit the user to "edit" and "paste" the contents of the buffer onto the command line. It would also be useful to store a stack of recent commands, allowing users to retain several frequently used commands. These commands could then be pasted on the command line and executed with two keystrokes whenever the user desired. Similar features are available on the DOS command line using the PF3 key.

(Parenthetically Speaking)

On a randomly selected page containing seven CLIPS rules there are 61 sets of parentheses. These represent 122 characters, 244 keystrokes, and about 8% of the characters on this particular page.

Since the CLIPS programmer may spend as much as 10% of his typing time addressing parentheses, some specialized functions to assist in this area might prove quite useful. The "action" menu in the CLIPS PC window is an excellent example of such a function. It will automatically format an "assert" or "retract" command for the user. This is a particularly useful function that would benefit even the experienced CLIPS programmer.

A similar function available in a CLIPS editor would be very useful, reducing the emphasis on typing parentheses and other symbols. An editor function that would place parentheses around a selected block of text would be helpful, too. This idea is closely related to the Command Storage Buffer and Editing issues addressed earlier.

On-line Help.

A strength of the CLIPS PC window user interface is the existence of its on-line help system. One feature of the help system that improves usability is its multi-level nature. Separate help is provided for the PC Window interface, CLIPS, and the help system itself. Since users ask questions at several different levels, this system is more likely to meet the user's needs in many situations.

The help system is well organized for letting the user "browse" through the information provided. This is a strategy that many users employ when learning a new system. By definition, however, a browsable help system generally does not respond well to ad-hoc requests. For example, the CLIPS user who desired information about the "retract" command would have to know (or find out) that this information resides under the menu items "using CLIPS" and "additional commands". A cross-referenced help system could provide help for both user strategies. Ideally, the browsable format would be retained and the system could also provide ad-hoc information in response to a command such as (help retract).

Remember that users often turn to on-line help for a quick answer to a specific question. By the time the CLIPS on-line help system is loaded and the user has mastered its tree structure, the original question may well have been forgotten. It is possible that the user may give up on the help system and turn to another source for assistance.

Dynamic Pull-down Menus and Mouse.

Application of a mouse and menu interface for CLIPS PC was a bold stride toward increasing the usability of the product. Selection by pointing and clicking with the mouse is almost always easier for the novice user. As users become more expert with a system, they tend to learn the keyboard equivalents for commands and spend less time using the menus and mouse.

The implementation of menus and mouse for CLIPS PC is based on the earliest level of technology. Compared to current products, the CLIPS window process is awkward and slow. Windows must be deliberately opened and closed, and selections are an active, very deliberate process. While errors may be less frequent under such conditions, user speed is drastically reduced. Professional programmers tend to prefer the potential for speed in their user interface as opposed to restrictive efforts intended to prevent errors. This would lead the CLIPS PC interface in the direction of the more dynamic mouse and menu technologies available today.

Conclusion

This paper has reviewed the usability of the CLIPS PC window system, pointing out some of its strengths and weaknesses and making some recommendations for possible improvements. It has suggested that the user interface in general move in the direction of a multi-window display. More important than any specific recommendation, however, is the suggestion that the CLIPS user interface be enhanced as its user community directs.

It should be pointed out that CLIPS platforms other than personal computers have had little or no attention paid to the attributes of their user interfaces. This paper has described some basic usability problems and solutions for one platform in an effort to promote the discussion of usability issues for all CLIPS implementations.