Command language vs. Menus or Both ?

J. Stelovsky*, H. Sugaya**

*Institut für Informatik, ETH Zürich, Switzerland
**Brown Boveri Research Center, Baden, Switzerland

Command languages have been widely used to describe the interaction between man and computer. Their syntax permits the expression of complex commands, but is considered a hindrance when it must be rembered. An alternative technique, menus and selection by pointing, has gained acceptance among users of interactive systems. Recently, this method has been further refined: icons and windows help represent and select data as well as command objects.

This paper describes a hybrid approach realized in the experimental system XS-2. In XS-2, the command language is combined with the menu selection by pointing, thus merging the expressive power of the former with the simplicity of the latter. In particular, the syntax of the command language is displayed and guides the user during the command input.

Key words and phrases: command grammar, command languages, user-interface models, menus and icons, human-computer interaction, universal commands.

1 Introduction

Command languages have been a major means of communication between man and computer. Some research efforts have focused on the formal specification of a command language and its use as a system specification and design language ([2], [3]). On the other hand, representation of commands as menus and their selection by pointing (e.g. with mouse) has captured the user's acceptance of interactive systems; icons and windows are used for representing data and command objects ([4], [5]).

The difference between the two approaches is the way the objects are identified. In the command language approach, command names and parameters must be typed in to be identified. Because the command language is based on a grammar, it permits the specification of complex, structured commands. On the other hand, a grammar imposes restrictions on possible user input. If the restrictions must be memorized, the syntax is perceived as an obstacle. In the menu approach, commands labeled by names or icons are displayed on the screen and used for input by pointing. In short, the command language is precise and expressive, but rigid if its grammar must be remembered. Conversely, menus are intuitive, but their use tends to oversimplify structure and grammar which are inherent to highly interactive applications.

These two approaches can be unified: we propose a hybrid approach where the command language and its syntax is used both as a system's output (presentation) language and as a user's input language. This approach incorporates the principles "see and point" as well as "remember and type": the commands can be accessed by name and the displayed commands can be selected by pointing. On the other hand, commands are structured with the help of "control commands" that permit expressing any command sequences based on regular expressions. Since the control commands are also displayed, the syntax of the command language is visible. The restrictions on the possible input need

not be remembered; the syntax is perceived as a guidance during the command input.

The hybrid approach has been realized in the interactive operating system XS-2 [6]. XS-2 grew out of a need for general, systematic and consistent human-computer interaction in a system with numerous application programs. It has evolved through an early experience with XS-1 [1] from which it has inherited and elaborated the key interaction concepts: Sites, Commands, Trails as well as Universal Commands. Data objects (Sites) and function objects (Commands) are represented in separate windows. A small set of Universal Commands is defined in the site and command windows. "Command grammar" based on Extended Backus-Naur Formalism (EBNF) defines command types that classify all commands: the commands in all application programs and the Universal Commands. These concepts constitute the user's model of the system.

2 Command language vs. Menus

Let us briefly discuss the trade-offs of the command language and menus.

1) Command language

The syntax of complex commands can be defined in the EBNF notation. Suppose we would like to define a command for the substitution of a string S_1 by a string S_2 in a source text. Each time the string S_1 was found, the user should decide whether he wants to substititute it or not. Furthermore, he should be able to perform or skip several substitutions in a row. Such a command can be defined in the following way:

Substitute = Find_S_1 Replace_S_2 { [n_times] (REPLACE | SKIP) } .

This definition tells us that the user has to input two strings (parameters "Find_S_1" and "Replace_S_2") and then repeatedly do the following: optionally type in a number (parameter "n_times") and decide between the parameters "REPLACE" and "SKIP" (e.g. by typing a key labeled "REPLACE" or "SKIP"). Notice that the command definition constitutes a syntactical rule for the legal sequences of user inputs.

In order to define a command language using EBNF notation, we can use the following command constructors:

Sequence: $C_1\ C_2 \ldots C_n$
Selection: $C_1 \mid C_2 \mid \ldots \mid C_n$
Option: $[\, C_1 \mid C_2 \mid \ldots \mid C_n \,]$
Repetition: $\{\, C_1 \mid C_2 \mid \ldots \mid C_n \,\}$

where C_i can be a terminal command or one of the command constructors. These command constructors correspond to the brackets used in our example and are equivalent to the constructors of regular expressions (without loss of generality with respect to the original notation).

Pros:

- Command language grammar can precisely express complex commands.
- Command language serves as part of the user's model of the system.
- Command language can be used for formal command specification ([2], [3]).

Cons:

- Command names must be memorized and typed.
- The syntactical rules are considered rigid, if they must be remembered.

2) Menus

Menus displayed on the screen indicate a current system state and the possible ways to change it: *what you see is what you get*. When an action is performed, a resulting state will be displayed again thus giving the user a feedback of the effect of his action: *what you see is what you've got*.

In Xerox Star [5], the menu technique is further elaborated by using icons to represent either a data or a function object. For example, to send a mail, a user points to a document (data) icon and then to a mailbox (function) icon. Icons are just another kind of labeling menu items. As pictograms, they are easily memorized. On the other hand, icons representing abstract objects are not intuitive. If icons are used only to classify objects, they also must be identified by name (consider, for instance, identical printers). Due to the limited size of the physical screen, only one level of menu or icons is usually displayed. In Xerox Star, for instance, the icons that contain other icons must be activated (i.e. opened) in order to reveal its contents. A "terminal" icon must be activated in order to see its commands; an option sheet window must be opened in order to see the parameter setting for a command. Thus, the menu approach accounts for tedious browsing through menu hierarchies.

Pros:

- Menus represent state information.
- Icons are easily recognized.

Cons:

- Icons may be imprecise; identification by names is still needed.
- Finding a command in deeper menu hierarchies is cumbersome.
- As in the command language, grammar needs to be remembered.

3 The conceptual model

The user interface model of XS-2 separates data and functions. A data object is called **site** and a function object **command**; sites and commands are organized as trees. XS-2 displays the sites and commands in separate windows (Fig. 1). Each site and command is identified by a name and classified by a type (written in italics in Fig. 1). A type corresponds to a symbol in a grammar. Grammars are used to control the structuring of sites and commands.

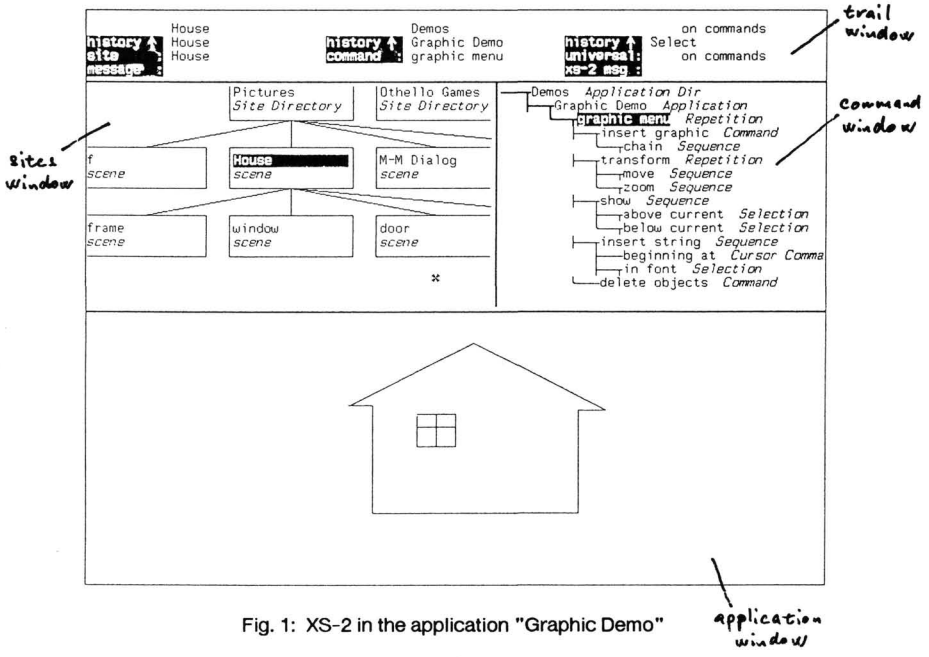

Fig. 1: XS-2 in the application "Graphic Demo"

A **site** is a representative of data attached to it. The **current site** represents the currently active data (highlighted in inverse video in Fig. 1). A **command** is a representative of an action function attached to it. Notice that the concept of **command** unifies applications, their commands and parameters. The **current command** represents the function which is currently being specified. Whenever a command is invoked and becomes the current command, its action function is applied to the current site. The resulting state changes are shown in the site and command windows. The sequence of state changes that have been executed in the past is not thrown away, but rather kept as history of the dialog. It is represented as a **trail** and displayed in another window.

A small set of commands commonly used on sites and commands is defined as **universal commands**. It is available at any time, even in the middle of a parameter sequence. The universal commands are of two kinds: view and motion.

A view command does not change the system's state, but only the portions of site and command tree visible on the screen. The user can change this representation and browse through the system and visualise its resources. To give an example, both tree representations shown in Figure 1 are available in the site and command windows. Browsing is done by scrolling. Since all sites and commands of all applications are permanently in the system database, scrolling allows the user to make any of the sites or commands visible. The visible sites and commands can be explained with another universal view command. Therefore, the user can get a "help" information for any site or command in the entire system at all times, i.e. even midst in a parameter sequence.

In contrast, a motion command changes the current state: the user can activate sites and execute commands. Each displayed site and command can be selected directly by pointing. Hence, the structures visible in site and command windows form hierarchical menus. At the same time, any site or command can be accessed by name. In particular, distant commands not shown in the menu can be activated this way, without having to go stepwise through menu hierarchies. In order to reach such a distant command, the system determines the shortest way between the old current command and the new one. If the new current command belongs to another application, the old application wil be unloaded and the new one loaded automatically. Thus, a user who remembers the command names can easily switch applications. XS-2 offers a variety of user-tailoring possibilities [7]; in particular, command names can be renamed. An experienced user can choose cryptic names in order to improve the access by name. Since universal commands can be entered by a function key, accessing a command by name requires only one additional keypress.

4 Sample Dialog

Let us illustrate the user's interaction with XS-2 with a sample dialog from the application "Graphic Demo". When we move the cursor into the command window and select the node "Graphic Demo", this application will be loaded. The command "graphic menu" below it becomes the current command; its subcommands are now displayed. The picture of the house is symbolized by the site "House" in the site window. This node is displayed in inverse video, indicating that it is the the current site. The application "Graphic Demo" displays its data in the application window (Fig. 1).

We now draw a picture for the site "door". We point the cursor to the box named "door" with the mouse, and select it with the mouse. The "door" becomes the current site and is now highlighted. The command "graphic menu" is of type *Repetition*; thus its subcommands "insert graphic", "transform" etc. can be repeatedly activated. Since we see also the command "chain" below "insert graphic", we can select it with the mouse.

The command "chain" is a *Sequence*; therefore the system traverses its subcommands in sequence. First we come to the command "starting point" where we are prompted for the point where the graphic

chain starts (Fig. 2).

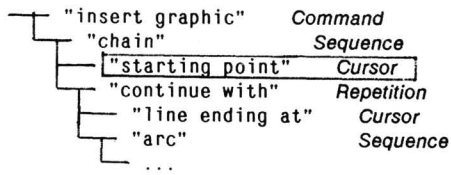

Fig. 2: At the current command "starting point"

The tree in Figure 2 can also be expressed in EBNF notation:

```
"insert graphic" = "starting point" {"line ending at" | ... } .
```

A door can be drawn with three consecutive lines: we point to the floor of the house with the cursor and enter the start point with the left mouse key. The system moves to the next command "continue with", where we can choose between drawing a line and an arc. We select "line ending at" to specify the end point. As soon as the cursor moves into the data window, the application helps us with a rubberbanded line. After we have entered the end point, the system returns to the *Repetition* "continue with", where we can repeatedly specify consecutive lines and arcs. Notice how our last steps are recorded in the trail window in Figure 3. When our drawing is finished, we select the command "insert graphic" once more. XS-2 leads us back to "graphic menu".

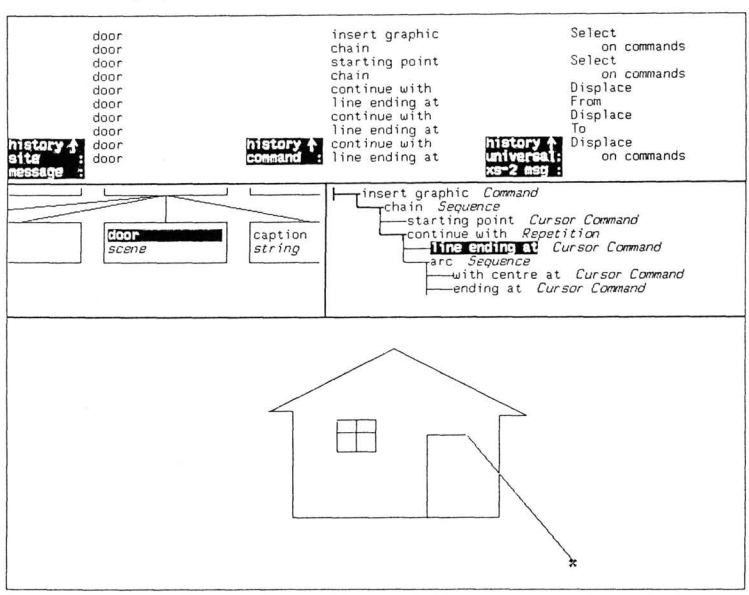

Fig. 3: Drawing polylines with the XS-2 application "Graphic Demo"

5 The command language with menus

The XS-2 command language consists of two levels: the top level is the universal commands and a lower level are the application commands. Both command levels are structured according to the same grammar.

1) Command Grammar

The XS-2 Command Grammar defines the types of commands and rules for their structuring.

Directory commands: The symbol *ApplicationDir* is the start symbol; it is used to organize *Application* programs.

ApplicationDir :: = { *ApplicationDir* | *Application* } .

The application consists of commands. Underneath an *Application* there is one control command that serves as an entry point.

Application :: = { *Sequence* | *Selection* | *Repetition* | *Option* } .

Control commands: The four control commands *Sequence*, *Selection*, *Repetition* and *Option* are the constructs necessary to define regular expressions. Their interpretation is the same as defined in Section 2. A control command defines how the subcommands will be activated. The subcommands of a *Sequence* are automatically activated in sequence. *Selection* requires the selection of exactly one subcommand. The selection of one subcommand in an *Option* is optional. In a *Repetition*, one of its subcommand can be selected repeatedly (including no selection). A subcommand is either a control command, a basic command or a dynamic command. Thus, the depth of a command structure is unbounded. In contrast to other command types, a control command does not invoke any action (i.e. application function)..

"control command" :: = { *Sequence* | *Selection* | *Repetition* | *Option* |
Command | *DynSelection* | *Text* | *Number* | *Cursor* } .

Basic commands: After the user specifies one of the basic commands *Text*, *Number*, *Cursor* or *Command*, the application's action procedure will be executed. *Text*, *Number* and *Cursor* supports the user's inputs from the keyboard or from the mouse. These three commands do not have any subcommands.

A command of type *Command*, when defined without subcommands, behaves as a command menu item. When defined with subcommands, it corresponds to an action whose effects must be further specified.

Command :: = [*Sequence* | *Selection* | *Repetition* | *Option*] .

The subcommands of a *Command* can be viewed as its parameters. In XS-2, however, a *Command's* subtree can contain other *Commands* with their own subtrees. Furthermore, the activation depends only on the type of a command and not on its position in the tree:

XS-2 unifies the concepts of commands and parameters.

Dynamic commands: The command types discussed above permit the definition of any command language based on regular expressions. But also context-sensitive (e.g. data-dependent) aspects can be considered: portions of the command tree can be hidden by the application program at run time and thus made inaccessible for the user. When a *DynSelection* is entered the application program is asked which subcommands of type *DynCommand* should be displayed. Upon selection, these sons will be hidden again.

Whenever a command type uniquely determines the next motion step, the traversal is performed automatically.

2) Application Commands and Universal Commands

The commands of all applications are permanently present in the command tree. Each command is classified by its type. A new application is created with the standard syntax-driven tool "Tree Editor". When the command tree is modified, the "Tree Editor" is driven by the Command Grammar.

As an example, the specification of the "Substitute" command from Section 2 can be directly translated into the following command tree:

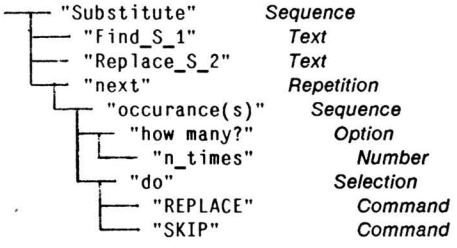

Fig. 4: Command structure for a "Substitute" command

Also the universal commands obey the XS-2 Command Grammar. Their trees are displayed as a menu upon request.

As we have seen, the XS-2 Command Grammar permits precise definition of complex command structures with the help of control commands. During the working session, the command tree is displayed as an automatically adjusted menu whose items can be selected by pointing. In particular, the control commands are also visible and can be selected. They tell the user how the commands can be specified:

> Command syntax is visible and perceived as an aid rather than a hindrance.

6 How to edit sets

Let us use the example of editing sets of numbers to show the variety of command structures that can be defined with the XS-2 Command Grammar. The editor should permit including and excluding one number as well as the range between two numbers. We show seven different command trees and discuss some of their advantages.

A straightforward version is shown in Figure 5. The two *Commands* "insert" and "delete" have identical subtrees except for the names. Each subtree is *Selection* between *Number* that represents a single element and *Sequence* of two other *Numbers* that define a range of consecutive elements.

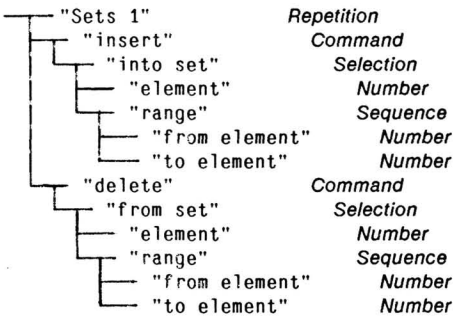

Fig. 5: Command tree for editing sets

It is likely that the user will keep inserting elements or ranges of elements. Under this assumption, we can improve the above commands. If the *Selection* is replaced by *Repetition* (Fig. 6) the user will not leave the command "insert" after an element or a range was specified. We omit the subtree of "delete" in the next three examples as it is equivalent to that of "insert".

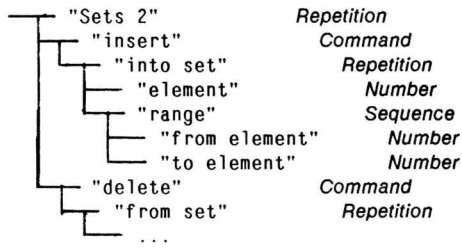

Fig. 6: Improved command tree for editing sets

We can also leave out the *Commands* "insert" and "delete" in Figures 5 and 6. The resulting trees are more shallow. Shallow structures improve the visibility of commands in the command window. On the other hand, deeper structures help casual users to specify their intentions.

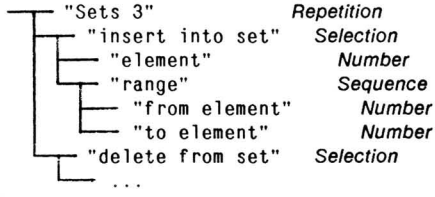

Fig. 7: Shallow alternative to Fig. 5

Let us now unify the specification of a single element and of the first element in a range (Fig. 8). The first son of the *Sequence* will be entered automatically. The second son has the type *Option*. Here, the user can enter the other limit of the range or select any other command shown in the command window. In the latter case, only one number will be inserted. This alternative is less intuitive than the previous examples, but saves an experienced user the choice between a single element and a range.

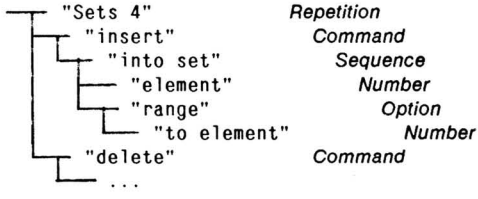

Fig. 8: Unified specification of an element and a range

In the previous examples, "insert" and "delete" had separate, but equivalent subtrees. Let us now combine them. We will prefix the definition of the "operands", i.e. the elements, by the definition of the "operation", i.e. a *Selection* between the *Commands* "insert" and "delete" (Fig. 9).

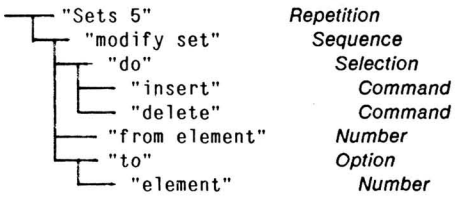

Fig. 9: Prefix form

This example can be easily converted to the postfix form, as shown in Figure 10. Here the *Selection* of the operation was moved to the end of the specification *Sequence*. In this command tree, the benefit of the *Option* is obvious: after the first element was entered, the user can point directly to one of the *Commands* "insert" or "delete".

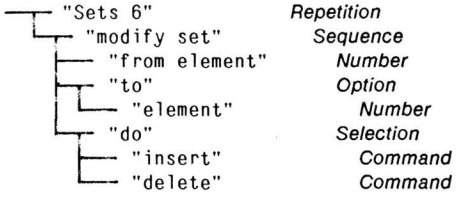

Fig. 10: Postfix form

Finally, we obtain the infix form or our last command tree by placing the *Selection* menu in between the specification of the operands (Fig. 11).

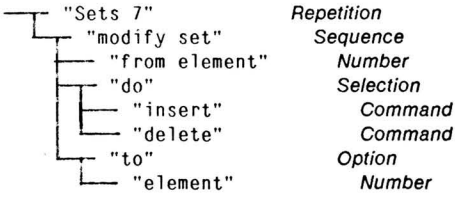

Fig. 11: Infix form

Still more alternatives are feasible. For instance, the subtrees in Figures 5, 6 and 7 can be merged and converted to a prefix or infix notation.

7 Concluding remarks

The XS-2 system is written in Modula-2 and runs on the personal computer Lilith [9] at the Institute for Informatics, ETH Zürich. It has then been ported to DEC's VAX-11 computer with a graphics terminal BitGraph (produced by BBN Inc.) at Brown Boweri Research Center, Baden [8].

The XS-2 approach inherits the expressive power of the command language and can describe a variety of command structures, in particular the prefix, postfix and infix notation. Since XS-2 displays data and functions as well as their structure as menus, the user can select them directly. The command language is used not only as part of the system's formal specification and as an input language but also as the system's presentation language. In XS-2, the command language along with its syntax is presented as menus and serves as a two-way means of communication.

Reisner [3] points out that

> "... in the long run better notational schemes need to be found or devised which reveal both structure and legal strings."

The XS-2 approach is a step towards this goal.

References

[1] Beretta, G., H. Burkhart, P. Fink, J. Nievergelt, J. Stelovsky, H. Sugaya, A. Ventura, and J. Weydert.
XS-1: An integrated interactive system and its kernel,
Proc. 6-th Int. Conf. on Software Engineering, Tokyo, 1982, 340-349.

[2] Moran, T.P.
The Command Language Grammar: a representation for the user interface of interactive computer systems,
Int. J. Man-Machine Studies, 15 (1981), 3-50.

[3] Reisner, P.
Formal Grammar and Human Factors Design of an Interactive Graphics System,
IEEE Trans. on Software Eng., SE-7, 2 (March 1981), 229-240.

[4] Shneiderman, B.
Direct Manipulation: A Step Beyond Programming Languages,
IEEE Computer Magazine, 16, 8 (August 1983), 57-69.

[5] Smith, D.C., C. Irby, R. Kimball, B. Verplank, E. Harslem.
Designing the Star User Interface.
BYTE, 7, 4 (April 1982), 242-282.

[6] Stelovsky, J.
XS-2: The user interface of an interactive operating system,
ETH Dissertation 7425, 1984.

[7] Stelovsky, J.:
User-Tailored Dialog - Just a Slogan?,
Proceedings of the International Computing Conference 1985,
Florence (March 1985), 345-351.

[8] Sugaya, H., J. Stelovsky, J. Nievergelt, and E.S. Biagioni.
XS-2: An Integrated interactive System.
Research Report, KLR 84-73C, Brown, Boveri & company, Ltd.,
Research Center, Baden.

[9] Wirth, N.
The Personal Computer Lilith.
Proc. 5-th Int. Conf. on Software Engineering,
San Diego, IEEE Computer Society Press (March 1981), 2-15.

Author:
Dr. Jan Stelovsky
Institut für Informatik
ETHZ
CH 8092 Zürich /Schweiz