User Interface Development for Existing Engineering Applications

Enhancement of Existing Engineering Software

Volume No. 6

by

J.A. Puckett, Professor

Chad Clancy, Research Assistant

Department of Civil and Architectural Engineering
University of Wyoming
Laramie, WY

March 1993

Disclaimer

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated under the sponsorship of the U.S. Department of Transportation, University Transportation Centers Program, in the interest of information exchange. The U.S. Government assumes no liability for the contents or use thereof.

PREFACE

This report is the sixth in a series involving the enhancement of existing engineering software. Previous reports addressed the interface of engineering application software with computer-aided drafting and applications that employed the techniques developed in this project. The focus of this report is on the user's interface with the engineering application. An interesting and novel method is described of an interface that can be developed in an extremely short time by engineers with no intimate familiarity with interface development. This report is intentionally brief and focuses on the methodology rather than the specifics of a particular computer-aided-software-engineering (CASE) tool or a particular application. Specifics will be of interest to the software engineer and can be obtained by contacting the authors. This report is written at the executive level in the hope that managers of transportation engineering operations will encourage development of software for their operations and the profession at large. As development tools become friendlier and more functional, in-house development/enhancement of sophisticated software for specialized tasks will become possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

There exists a large body of software in the engineering design professions that is routinely used, well tested and trusted, and is often very specialized. This software is often developed and maintained in-house, or by those quite familiar with the engineering aspects of the particular code but often unfamiliar with the most current advancements and methodologies required to make the software user friendly. It is common for a windows-based user interface to take as much or more development time than the engineering application. The development of modern interfaces—X-windows, Microsoft Windows, Apple Macintosh OS—require significant time and specialized expertise, but such resources are often not available for small in-house development projects and/or familiar to developers writing specialized engineering applications.

The principal objective of this report is to describe a methodology for the development of user interfaces that is appropriate for existing engineering software. This methodology is appropriate for applications that are specialized and may not command the market required for extensive broad-based commercial development. The authors are most familiar with applications used by the bridge engineering community and this is reflected in the applications developed to date. Application of the general concepts described are broad and appropriate for many applications other than structural design.

DEVELOPMENT OBJECTIVES

The two principal objectives for this project were to:

Formulate a methodology for the development of user interfaces for existing engineering software.

Formulate a methodology to interface existing engineering software with CAD.

The first objective is the focus of this report. Methodology is described for development of user interfaces for engineering applications.

The second objective was addressed in two MPC reports titled, *Linking Computer Aided Engineering Procedures with Computer Drafting* (Puckett and Clancy, 1991) and *UWGRAPH Computer Aided Design and Drafting Library* (Puckett and Clancy, 1991a, 1991b). The first report was a short description of the architecture, application development, and a FORTRAN library for creating CAD files directly from existing procedures. The second was programmer documentation for the library. This information is not reiterated here.

The first objective was initially addressed by focusing on the functional requirements rather than with the implementation of computer/software platforms. Several functional requirements for the system and its development are outlined below with a brief discussion:

Consistent Interface Behavior: The architecture is the same for all applications. This required a library of functions to be written which offers the same "look and feel" for all applications.

Menu-based operation: The system is able to accommodate many input screens (over one hundred) and a menu system is the best method for accessing screens.

Large number of input screens: Many important applications in the transportation/structural engineering fields require several commands for the data definition.

CASE Development: Hundreds of input screens are planned or have been developed, therefore a computer-aided-software engineering (CASE) tool is required to draft the screens. It is impractical, if not impossible, to create all these screens with a low-level library such as X-lib (Nye, 1990).

Graphical orientation: Engineering data are often best defined with the use of a graphical representation of the system being described. Dimension data are examples.

On-line help: The help system incorporates all the information contained in the hard-copy documentation. Access to this information is context sensitive and/or uses systematic indexing and search procedures. For example, keyword searching is employed.

Modular design: The user interface is completely separate from the application that performs the engineering calculation. Communication is by way of data files and/or commands created by the interface. Separation of data definition and subsequent use of data allow the development and maintenance of the user interface to proceed with little collaboration by the engineering application developer. This simplifies development and maintenance and allows each type of developer (interface and engineering) to focus on their area of expertise. Of course, one individual may be the developer of both interface and engineering applications.

Minimal or no code development: Using current development tools, an interface is developed with minimal amount of new code. The interface developer is literally able to draft the interface with little or no effort in a high-level language.

Opportunities for network computing: The system accommodates the use of a network, allowing the machine where the interface resides to communicate with another machine where the engineering application resides. This function allows the interface

machine (graphical device) to pass data to the application machine that may not be graphically oriented. The two machines may be the same machine.

Portability of methodology: The architecture is "portable" to different computer platforms. This does not imply portability of code or the library, but the methodology developed is the same across operating environments.

SYSTEM ARCHITECTURE AND DESIGN

The system architecture has three distinctive parts as illustrated in Figure 1 - the interface code, the application code, and the graphics application. Typically, the communication between all these programs is with ASCII files. The information is initiated by using the interface code. The interface produces a data file and/or interactive commands to the engineering application. These commands are the same commands typically entered by typing or processing batch data files. The application passes data to a graphics application that creates a CAD file which is interpreted by the CAD system. Such files include DXF (Autocad) and IGDS (Intergraph) formats. The architecture of the interface code is the focus of this report. The engineering application is the responsibility of the developer and is not described. The graphics applications incorporating UWGRAPH are described in detail elsewhere (Puckett and Clancy, 1991a).

The interface code architecture is illustrated in Figure 2. The core of the system is a function library that performs initialization of the system, interprets menu commands, displays input screens, and interprets the internal commands from the input screens, and finally issues commands to the engineering application. The menus, input screens, and help files are contained in an abstraction termed the application resource. Application resources are different for each application and are created using a CASE tool that allows the developer to literally draft the input screens and menus. Intergraph I/FORMS (Intergraph 1992) and Borland C++ with Application Framework (Borland 1992) were used in this project. The particular CASE tool used is an important issue in the development and maintenance of the software, but is not directly relevant to the discussion of the system architecture and implementation.

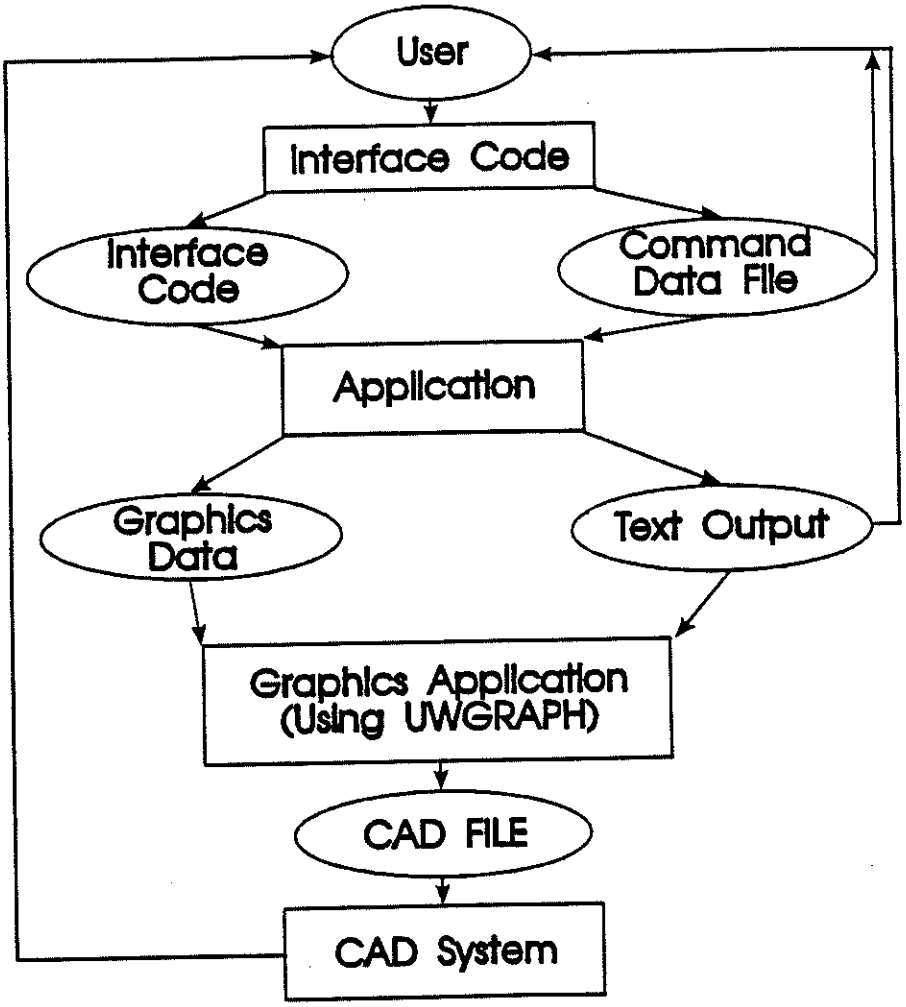

Figure 1. System Architecture

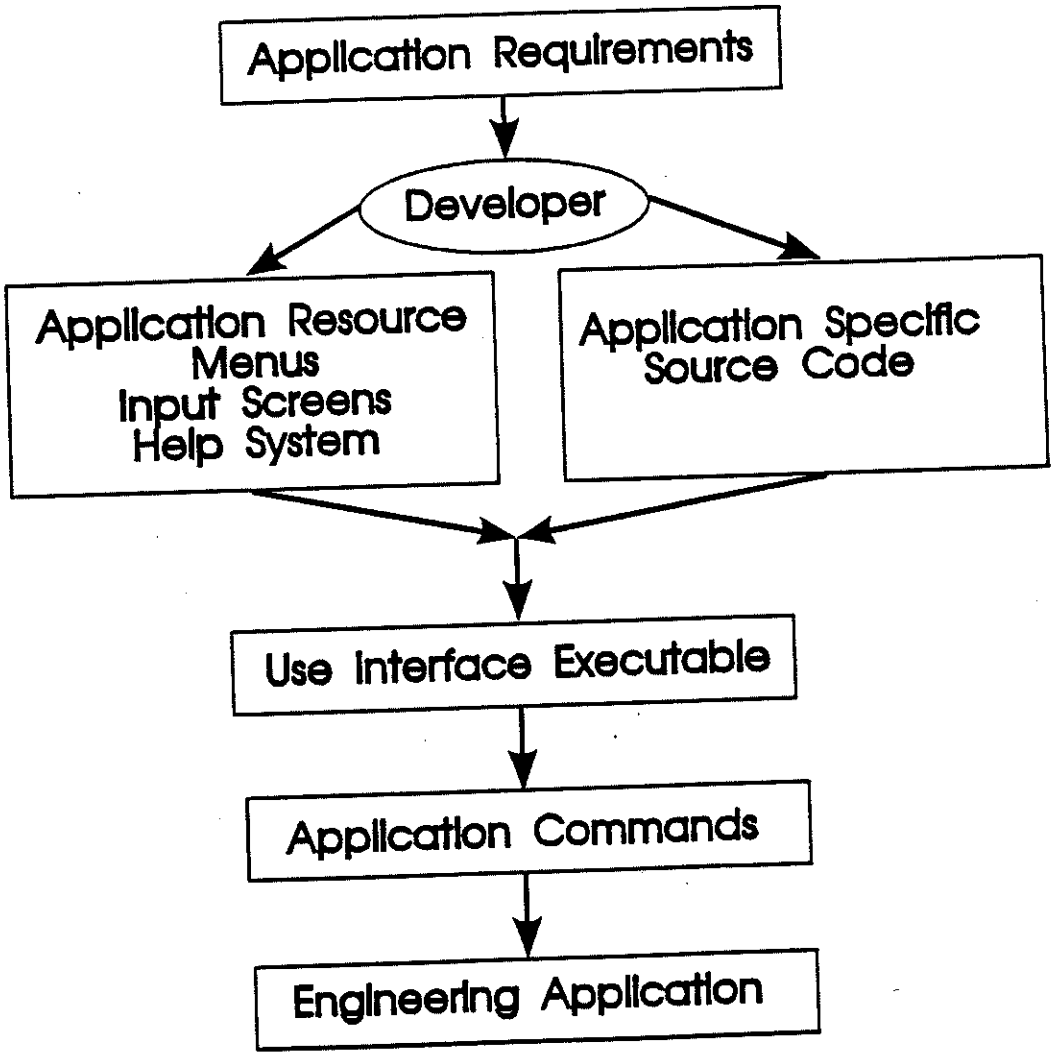

Figure 2. Architecture of User Interface

The interface functions are organized in a library and are independent of the application. The key to this set of functions is that the library must accommodate *all* applications and *all* input screens. This important aspect requires that each application resource contain the screen information necessary for resource definitions for the user interface, *and* all the information required for the interface library to produce the command data to the application. Because all engineering application data are unique to the application, the application command data must be defined during the screen drafting process and be contained in the application resource rather than included in the interface code. Most CASE tools are *not* designed to directly support this functionality.

The application commands are defined by mapping the user input fields to a C format string. For this project, the format was stored in a "hidden field" default value in the case of I/FORMS and in the string description attribute in Borland C++. Both CASE tools permit runtime access to these data and the process is totally transparent to the end user. The code proceeds sequentially as described below:

1. The library addresses the application resource to obtain the screen descriptions and uses these data to display the screen.
2. The library controls the user interaction.
3. The user enters the data in a manner consistent for all screens.
4. Upon a written request or an acceptance, the library addresses the hidden format, data, check bounds, etc. and maps the user data to the format and issues the application command to application or data file.

IMPLEMENTATION SPECIFICS

Application of CASE

Many CASE tools are available for application development under Microsoft Windows (Microsoft 1992), Macintosh OS (Apple Computer 1992) and X-Windows (OSF 1991, Nye 1990). As expected, the CASE-tool functionality varies greatly, sometimes requiring the programmer to handle the important but mundane tasks by writing code in a relatively hard-coded manner. Such tasks include checking default values, checking minimum and maximum bounds, illustrating relationships between data, maintaining the order of field access, and controlling context-sensitive help. Some tools handle much of this functionality at the screen drafting stage and store these data in the application resource while other tools require that the interface application handle these issues by writing code specific to a particular application. Without elaborating on the specifics of various CASE tools, it is justified to state that the way tools address these issues is extremely important and varies widely. Because of these variations, some objects require enhancement by extension of the objects' attributes. This is described in the next section.

Attribute Extensions

In the two CASE tools used to date, both were extremely useful but are very different in the level of developer support. The important CASE feature that the present methodology must possess is the ability to store string data in the application resource and extract that data at runtime. With this function, the interface developer can define a host of attributes that may not be directly supported by the CASE tool and associated library. For example, a "string attribute" associated with each field could contain the following string:

type = float; max = 10; min = 0; message = "data exceeds bounds"; default = 1.5;

where

type is the data type indicator,

max is the maximum value for the field entry,

min is the minimum value for the field entry,

message is the string to be displayed upon error detection, and

default is the value initially displayed.

A "screen attribute" which controls the output of all fields within a screen might contain the format string:

format = "command string %s, %d, %f\n next command %d, %d \n"

format is the string passed to the library that is used in format editing the application command data.

The attribute extensions given are examples, and clearly, more attributes can be added as necessary. This approach allows the interface library to effectively handle attributes that are not directly supported by the CASE tool. Furthermore, the command format string is available to the interface library and this is a necessity in order for a single library to support interfaces for multiple applications that require unique commands.

Development Process

Interface development begins by planning the appearance of each data input screen. This is facilitated by the engineering application documentation. One or more commands can be combined on one data input screen. Next, the screens are organized into groups and mapped to menus. For example, a command for uniform dead load would be located on the *loads:dead:uniform* menu. The developer then drafts the input screens

and creates the menu using the CASE tool. All extended attributes are included in the input screen at this time. Any application-dependent code is written. Such code is typically minimal and usually deals with the grouping commands to be displayed in a logical order. For example, perhaps it is logical to display the concentrated load screen immediately after the uniform load screen, but these loads are defined on separate input screens. Another case for system dependent code is to display input screens for an entire type of input file or system. For example, display all the screens needed to define a multi-span steel plate girder and associated loading. Those displayed many represent only a fraction of the total possibilities, therefore input is greatly simplified by leading the user through the appropriate data entry.

APPLICATIONS

Several applications have been developed with the methodology described. The applications, their function, and other statistics are given in Table 1.

Table 1. Applications Developed to Date

Application	Function	Number of Screens[1]	Engineer-days Interface Development[2]
BRASS-Girder	Bridge rating and analysis of structural systems	99	45
BRASS-Bearing	Bridge Bearing Design and Analysis	5	5
BRASS-Dist	Load distribution in Slab-Girder Bridges	16	5
BRASS-Culvert	Culvert Design	14	7

1. All applications have been developed with both Intergraph I/FORMS and Borland C++.

2. Times are approximate. The development of the library was performed at the same time and this made time and effort quantification difficult.

A typical input screen from BRASS-Girder is illustrated in Figure 3. This screen is functioning under Microsoft Windows and has all the functionality of a windows application, such an iconification, cut/copy, and paste across applications, etc. An editor is supported in the interface library which allows the user to edit command data after, or in conjunction with, the input screens. The edit screen is illustrated in Figure 4. Note support of features such find, replace, cut, copy, paste, over type, insert, and delete. On-line help is available by way of table of contents, index, and/or context-sensitive activation. An example of an on-line help page is illustrated in Figure 5.

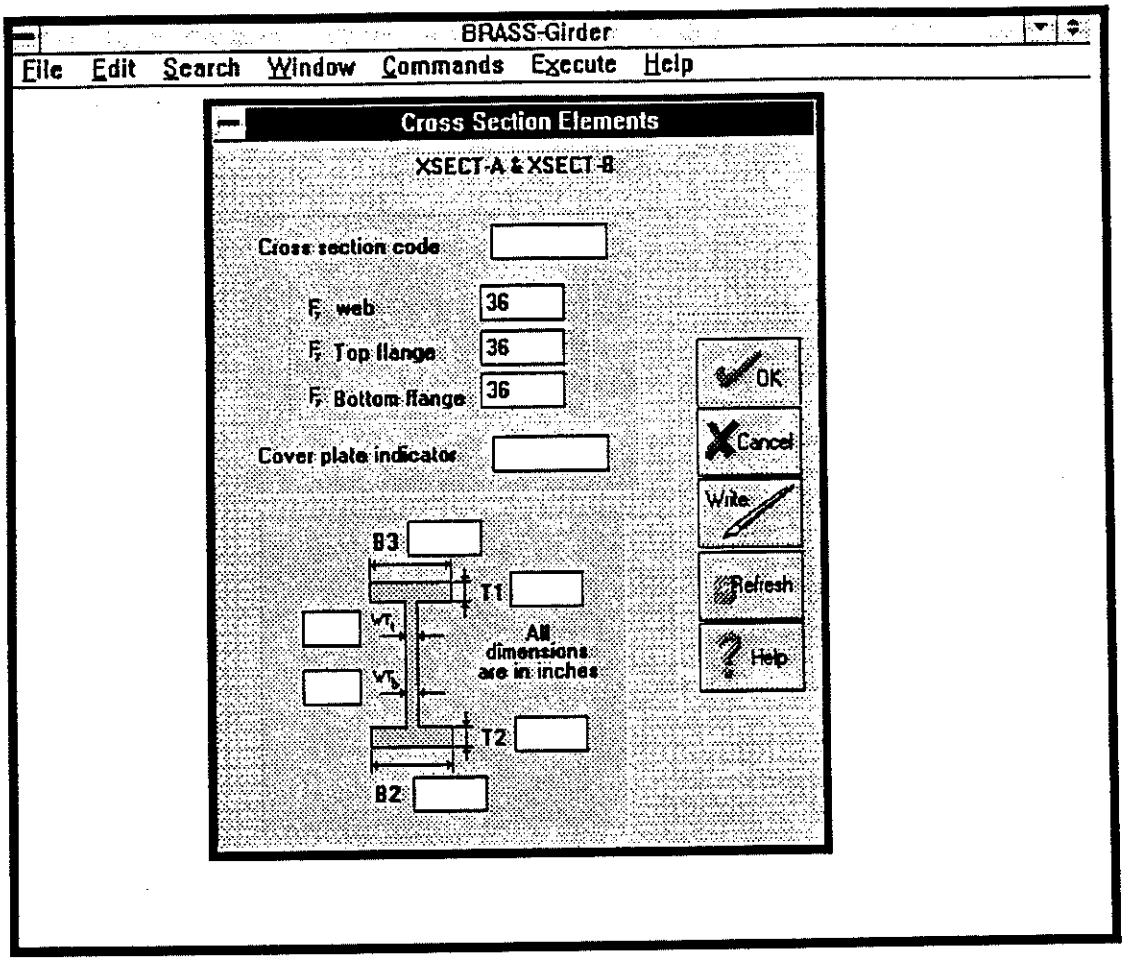

Figure 3. Typical BRASS Dialog Window

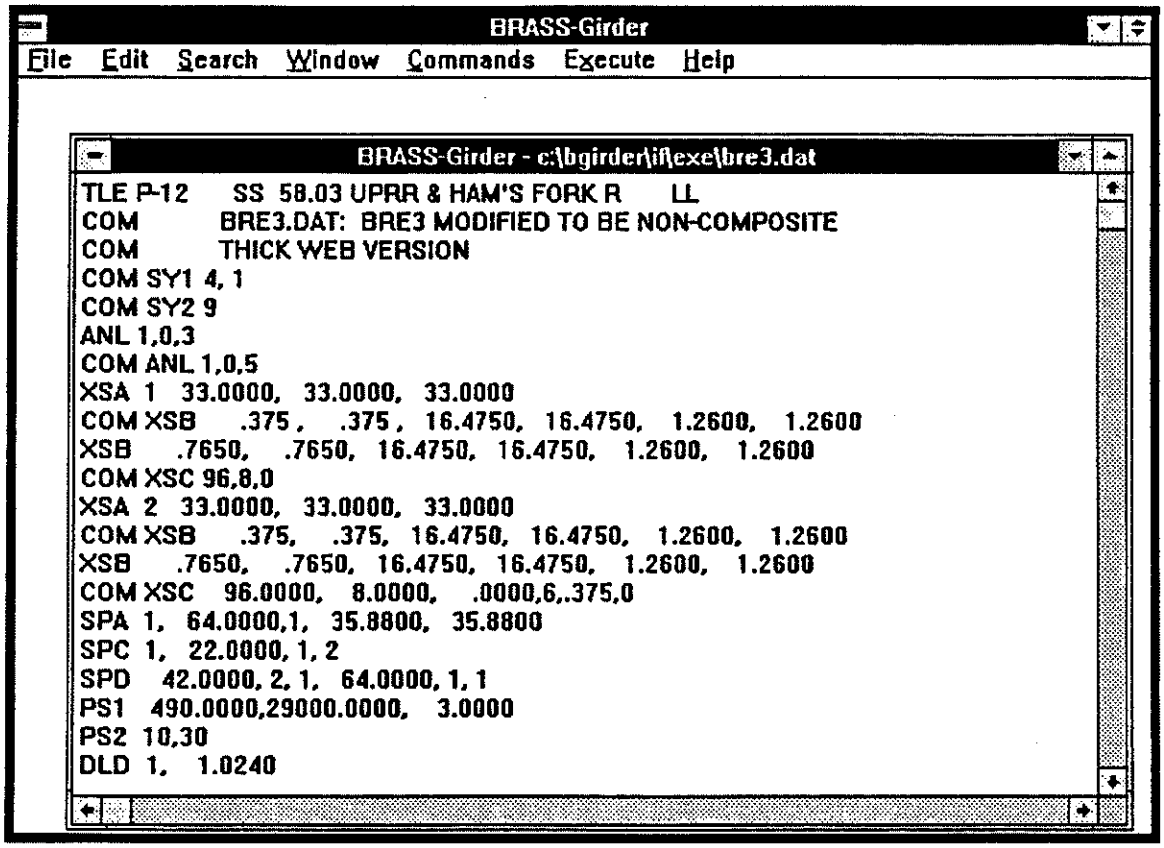

Figure 4. BRASS-EDITOR

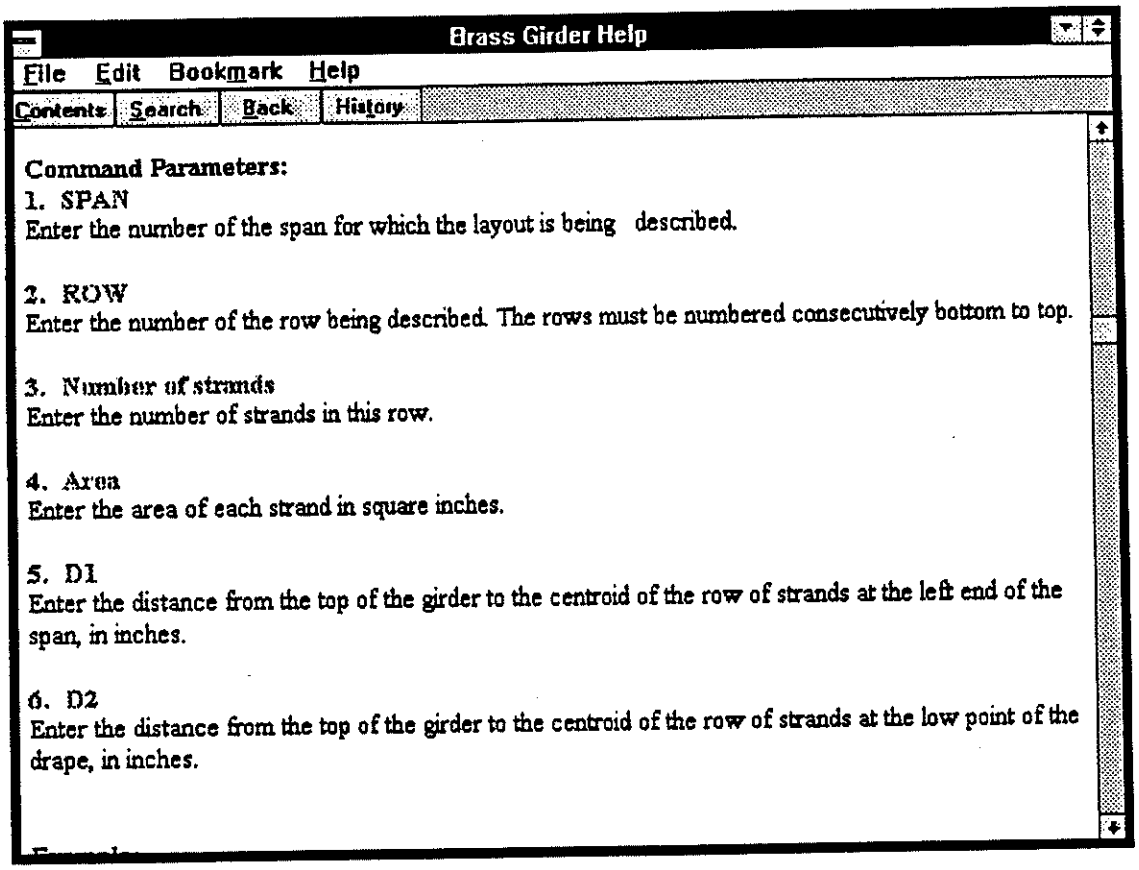

Figure 5. Typical Help Reference

SUMMARY

A methodology for the development of user interfaces for existing engineering applications is described. The architecture is described and key features are elaborated. A detailed description of adding attributes—and hence functionality to CASE-tool support for development—is given. Applications have been developed using this methodology and these applications have been used in a production environment.

REFERENCES

Puckett, J.A. and Clancy, C., *Linking Computer Aided Engineering Procedures with Computer Drafting*, Final Report, Mountain Plains Consortium, No. 007, Vol. No. 1, 1991.

Puckett, J.A. and Clancy, C., *UWGRAPH Computer Aided Design and Drafting Library,* Final Report, Mountain Plains Consortium, No. 007, Vol. No. 2, 1991.

Nye, Adrian, *X-lib Programming Manual for Version 11*, Release 4, O'Reilly and Assoc. Sebastopol, CA, 1990.

Intergraph Corp., *I/FORMS Programmer's Reference Manual*, Huntsville, AL, 1991.

Borland International, Inc., *Borland C++ with Application Framework*, Programmer's Guide, Version 3.0, Scotts Valley, CA, 1992.

Microsoft Corp, *Microsoft Windows User's Guide*, Version 3.0, Redmond, WA, 1990.

Apple Computer, *Macintosh Reference 7*, Palo Alto, CA, 1991.

Open Software Foundation, *OSF/Motiff Rev. 1.1*, Prentice Hall, Englewood, NJ, 1991.