

A Prototype Assistance Manager
for the
Simulation Model Development Environment
by
Valerie L. Frankel

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Computer Science

APPROVED:

Osman Balci, Chairman
Department of Computer Science

Richard E. Nance
Department of Computer
Science

H. Rex Hartson
Department of Computer
Science

July 27, 1987
Blacksburg, Virginia

**A Prototype Assistance Manager
for the
Simulation Model Development Environment**

by

Valerie L. Frankel

Osman Balci, Chairman
Computer Science
(ABSTRACT)

The Assistance Manager, one of the tools of the Simulation Model Development Environment (SMDE), is required to provide assistance to a user during the process of model development. This thesis describes the research effort to prototype the SMDE Assistance Manager.

Requirements are set forth and a design is established for the Assistance Manager prototype. The implementation is described, and the Assistance Manager is shown to provide a highly flexible interface between the user and the database of assistance information.

Assessment criteria are established, and the prototype is evaluated. Results of the evaluation indicate that the Assistance Manager incorporates the characteristics considered desirable in online assistance systems, and serves as a basis for future enhancement and development.

TABLE OF CONTENTS

1. 0	<u>INTRODUCTION</u>	1
1.1	The SMDE Research Project	1
1.1.1	SMDE Architecture	1
1.1.2	The SMDE Toolset	4
1.2	Problem Statement	6
2.0	<u>LITERATURE REVIEW AND SCOPE OF RESEARCH</u>	8
2.1	Background	8
2.2	Classification of Help Systems	10
2.3	Issues in User Interface and Help System Design	14
2.3.1	How Many Levels of Help Should Be Available?	16
2.3.2	Should the User or the System Decide When Help is Required?	17
2.3.3	What Factors Should Be Considered in the Textual Composition of Online Assistance?	17
2.3.4	How Can Online Assistance be Provided Easily and Efficiently by Programmers?	18
2.4	Survey of Some Existing Help Systems	19
2.4.1	BROWSE	19
2.4.2	AT&T Unix HELP Facility	19
2.4.3	ICNHELP	20
2.4.4	SIGMA	20
2.4.5	ZOG	21
2.4.6	TNT	21
2.4.7	SARA	21
2.4.8	HELP: A Question Answering System	22
2.4.9	ACRONYM	22
2.4.10	DOMAIN/DELPHI	22
2.4.11	UC: The Unix Consultant	23
3.0	<u>DESIGN AND IMPLEMENTATION OF THE PROTOTYPE</u>	24
3.1	Motivation: Factors in the Design of the Assistance Manager	24

3.1.1	Characteristics of Effective Online Assistance	25
3.1.2	Assistance Manager Design Objectives and Requirements	27
3.1.2.1	Design Objectives	27
3.1.2.2	Functional Requirements	30
3.1.3	Prototyping as a Design Strategy	31
3.2	Hardware/Software Environment	32
3.3	Components of the Assistance Manager	34
3.3.1	Introduction to the MDE	35
3.3.2	Tutorial	38
3.3.3	Glossary	43
3.3.4	Local (Tool-Specific) Help	47
3.3.5	Comment Facility	51
3.3.6	Programmer Assistance	51
3.4	Implementation Considerations	56
4.0	<u>EVALUATION OF THE PROTOTYPE</u>	59
4.1	Assessment Criteria	59
4.2	Evaluation of the Criteria	59
4.3	Design Alternatives	65
4.4	Design Limitations	67
4.5	The Choice of INGRES as a Database Management System	68
5.0	<u>SUMMARY AND CONCLUSIONS</u>	70
5.1	Summary	70
5.2	Recommendations for Future Research	71
5.3	Conclusions	73
	<u>BIBLIOGRAPHY</u>	75

APPENDIX 1 75

APPENDIX 2 83

VITA 84

LIST OF FIGURES

Figure 1.1.1.1	SMDE Architecture	3
Figure 3.3.1	Assistance Manager Tool Menu	36
Figure 3.3.2	Invoking the Assistance Manager through an SMDE Tool	37
Figure 3.3.3.1	The Introduction to the SMDE Component of the Assistance Manager	39
Figure 3.3.3.2	Searching for a phrase within the Introduc- tion to the SMDE	40
Figure 3.3.2.1	The Tutorial Component of the Assistance Manager	42
Figure 3.3.3.1	Glossary Main Menu (left) and Glossary Lookup Menu (right)	44
Figure 3.3.3.2	Glossary Browse Menu	46
Figure 3.3.4.1	Example Demonstrating Help Frame Feature of Local Help	49
Figure 3.3.4.2	Example Demonstrating the "Explain" Feature of Local Help	50
Figure 3.3.5.1	Comment Form (right)	52
Figure 3.3.6.1	Programmer Assistance Main Menu (right)	53
Figure 3.3.6.2	Example of Interaction to Add Documents to the Documents Directory	55
Figure 3.3.6.3	Quick Reference Guide Window (right)	62

ACKNOWLEDGMENTS

The author wishes to acknowledge her chairman, Dr. Osman Balci, for his invaluable assistance and total commitment to the research project, and in particular, to his role as advisor and professor. Acknowledgment is also extended to Dr. Richard E. Nance for many valuable insights and critiques of the work described in this thesis. In addition, I wish to thank Bob Moose, Lynne Barger, and Joe Derrick for the technical support they provided as members of the MDE Research Project; Dr. Rex Hartson for serving on my committee; and the entire faculty and staff of the Computer Science Department for the productive years I spent in close contact with them.

This thesis is dedicated to my husband, Jay, who has always supported me completely and never doubted my ability to persist despite what seemed at times to be insurmountable odds. His love and patience have been the guiding forces of my life.

1.0 INTRODUCTION

1.1 The SMDE Research Project

The use of computer-based models is an activity used to solve problems in all areas of business, government, industry, and the military. As the problems grow larger, become more complex, and require accurate solutions more rapidly than ever before, the model development process itself requires computer assistance throughout its entire life cycle [Balci 1986].

Research is ongoing at Virginia Tech to prototype a Simulation Model Development Environment (SMDE) which would automate the model development process, thereby reducing the cost of development and increasing the quality of resulting models.

The objectives of the SMDE, as stated in [Balci 1986], are to

- (1) offer cost-effective integrated support continuously throughout the entire life cycle of model development,
- (2) improve the model quality by effectively assisting in the quality assurance of the model,
- (3) significantly increase the efficiency and productivity of the project team, and
- (4) substantially decrease the model development time.

1.1.1 SMDE Architecture

The four-layer architecture of the SMDE is depicted in

Fig. 1. The hardware and operating system reside at Layer 0. For the SMDE prototype, a Sun workstation running the Unix* operating system provides the Layer 0 base.

Layer 1, the Kernel Model Development Environment (KMDE), integrates the SMDE tools into the programming environment. The KMDE provides databases, communication and run-time support functions, and a kernel interface.

Layer 2, the Minimal Model Development Environment (MMDE), provides a comprehensive toolset considered minimal for the development and execution of a model. It is basic in the sense that the set of tools enables modelers to work within the bounds of the MMDE without significant inconvenience, and general in the sense that the toolset is generically applicable to all types of abstract modeling tasks.

The outermost layer, Layer 3, is the highest layer of the environment which is based upon a particular MMDE. In addition to the toolset of the MMDE, it incorporates tools that support specific applications and are of special interest within a particular project.

An SMDE tool is integrated with other tools and the programming environment through the kernel interface. The provision for this integration is indicated in Fig. 1 by the opening between Project Manager and Text Editor.

*Unix is a trademark of AT&T Bell Laboratories

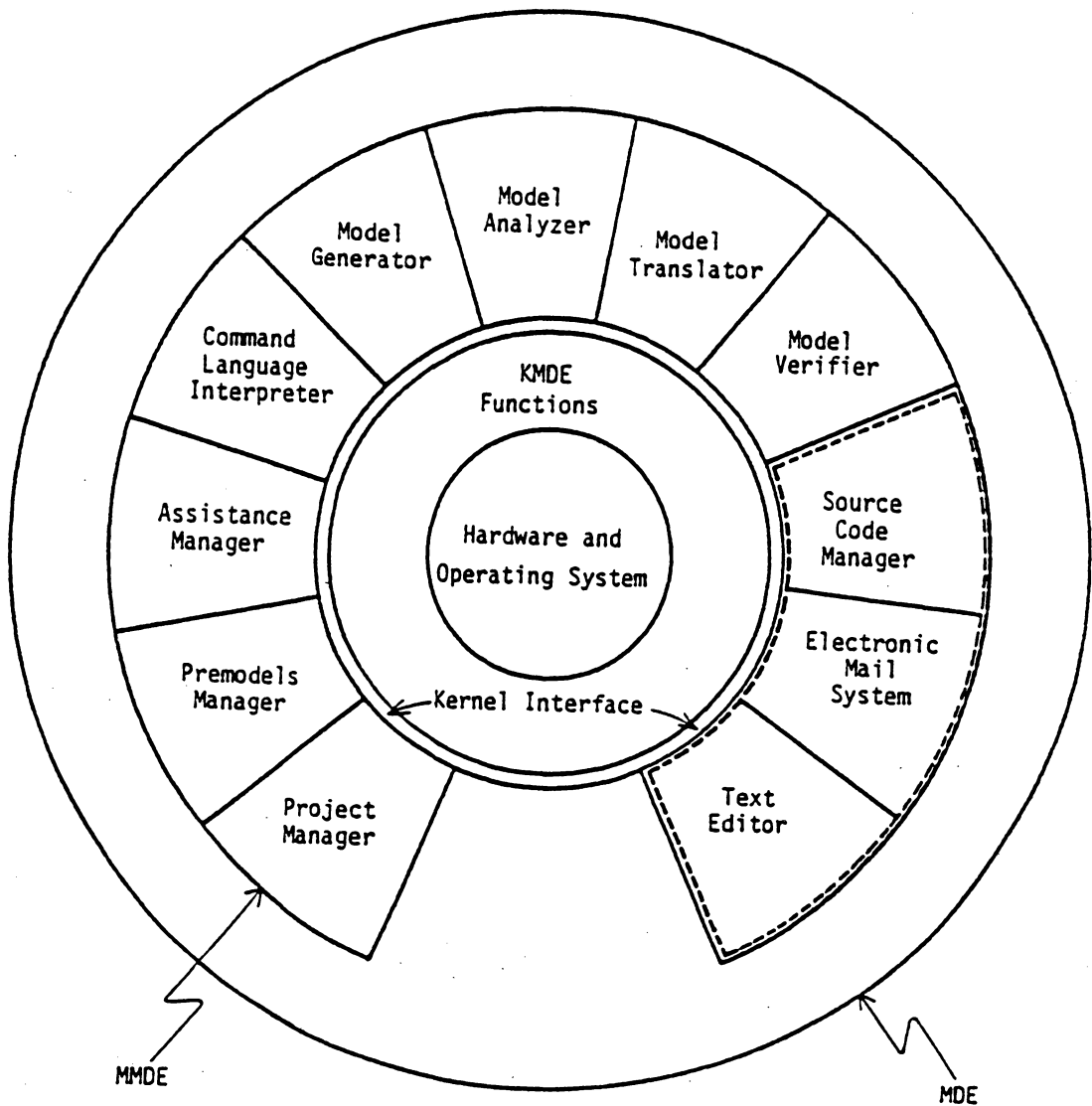


Fig. 1.1.1.1 The SMDE Architecture

1.1.2 The SMDE Toolset

The SMDE provides a comprehensive set of tools that is considered minimal for the development and execution of a model. A brief description of these tools is given below.

The **Project Manager** responds to queries about project status and due dates, triggers messages and reminders, records the progress of the project, and administers the project database.

The **Premodels Manager** administers the premodels database containing earlier developed and tested models, submodels or model components, and provides information on previous modeling projects.

The **Assistance Manager** administers the assistance database, provides tutorial assistance, gives definitions of technical terms, and explains details of tool usage.

The **Command Language Interpreter** is capable of invoking all SMDE tools through an extensible and human engineered interface.

The **Model Generator** supports the modeling process by guiding a modeler through model definition, model specification, and model documentation. It also provides assistance in model qualification.

The **Model Analyzer** diagnoses the model specification created by the model generator and assists in verification of the communicative model.

The **Model Translator** translates the model specification into an executable code after the quality of the specification is assured by the model analyzer.

The **Model Verifier** assists in verification of the programmed model by creating a cross-reference map, supplying dynamic analysis tools such as traces, breaks, and system snapshots, and incorporating diagnostic measures within the source program.

The **Source Code Manager**, assumed to be provided within or by the programming environment, translates the source code into a machine representation and performs its execution.

The **Electronic Mail System**, also assumed to be provided by the programming environment, is necessary for communication among the people involved in the project.

The **Text Editor** is expected to be included within the programming environment. It is required for general text preparation, including technical reports, user manuals, and system documentation.

The multiplicity of the SMDE tools forces increased attention to be focused on a uniform, flexible assistance system that is responsive to user errors and assistance requests. The Assistance Manager is required to address this need, and the research described in this thesis directly concerns the design and implementation of a prototype Assistance Manager.

1.2 Problem Statement

The Assistance Manager, one of the essential components of the Simulation Model Development Environment, is required to provide help on the use of an SMDE tool, and to perform other assistance tasks for the modeler or tool developer [Balci 1986].

Because the SMDE is a general purpose environment that can support a variety of modeling and simulation applications, it is not unreasonable to expect that it may be used by people with widely different backgrounds and expertise. It is even conceivable that the SMDE may be used by an analyst who has no prior computer experience at all. For a modeler to work productively and confidently within the SMDE, great care must be given to design an interface that is integrated with an online assistance system. This system, to be effective, is required to be constantly available, easy to use, and sensitive to the user's current state. Furthermore, these requirements must be met by a help system that does not dominate the user's interactions within the environment. It must be flexible enough to accommodate varying levels of user expertise, and should be accessible from any tool in the SMDE. Because the SMDE is an open-ended environment, the help system must be general and flexible enough to be easily incorporated into existing tools, as well as included with new tools which

may be added to the environment later.

No existing help system can meet all these requirements. Current help systems available for general use are add-on systems designed to operate independently of the system under study. They lack the flexibility and context sensitivity necessary for use with the SMDE.

This thesis is put forth to develop a prototype Assistance Manager, based on sound principles of human-computer interaction and help system design, which meets the requirements listed above.

2.0 LITERATURE REVIEW AND SCOPE OF RESEARCH

2.1 Background

A help system, as referred to in this thesis, is designed to aid a user in using some software facility. This aid may be offered in the form of command assistance, error recovery, online documentation, or computer-aided instruction. The assistance may be separate from the system on which help is offered, or it may be integrated closely with the system architecture.

Currently, the standard practice is to include some form of online assistance with an interactive system. From the user's point of view, the major differences in these help systems are in the kind of help offered, and the mechanisms used to access the information.

Many researchers who characterize their work as "online help" consider their design in terms of users recovering from errors. These systems are concerned with context sensitivity; i.e., how the system can find out enough about what the user was doing to explain the problem and offer advice. People who characterize their work as "online documentation" typically think of making reference material available to the user on request [Walker 1986].

When the amount and depth of information needed to start using a system is more than can be addressed by a help system, online tutors can be used. Tutors have the advan-

tage of offering a new user a safe, restricted environment where he can learn and subsequently test his knowledge. A familiar example of an online tutorial is the "learn" command available on many Unix systems [Kernighan and Pike 1984].

Most older help systems, and some current ones, are add-on systems: software facilities that are separate and in addition to the system being used. Easily maintained and updated, the major disadvantage is that the user's current task must be interrupted while the user searches through online files to find the particular piece of information he seeks. Such systems are little more than online reference manuals.

With the advent of personal computers, computing became accessible to a diverse population. Interactive systems became more widespread and no longer could intuition and ad hoc methods suffice to design help systems. Help began to be integrated: developed around -- and not in spite of -- the software system itself. Examples of integrated help systems are systems that base their user interface on menus with well-described choices [Bergman 1985].

At first, this "integrated" help was nothing more than the incorporation of "user-friendly" features. Indeed, much early research on computer help systems dealt with user interface design [Mozeico 1982; Relles and Price 1981;

Robertson et al. 1981; Relles 1979; Sondheimer and Relles 1982].

Obviously, the two domains overlap a great deal. A well-designed interface can alleviate the need for an outside help command. However, recent research [Bergman 1985; Bramwell 1983; Butler and Kennedy 1985; Estrin et al. 1986; Nakatani 1986] tends to treat help systems as distinct from user interface issues.

2.2 Classification of Help Systems

A lack of consistent terminology has hampered any universally accepted classification of help systems. Fenchel [1982] suggests as a first step in developing a taxonomy of interactive assistance a classification into four distinct types of assistance information:

- 1) expert systems - systems that attempt to give the advice that would typically be given by an expert human,
- 2) tutorials - lesson-based assistance demonstrating the use of the software system/tool. The emphasis of a tutorial is on teaching, not on the user's present situation.
- 3) peer assistance - a collection of anecdotal information provided by previous users of a software system which reports bugs, describes special features, and explains functionality of a system.

This may consist of "how-to" information, "cheat-sheets", and bulletin-board style notices organized in a single location to enable browsing by users.

- 4) contextual/command help - help that enables a user to request a specification of commands that are available in the current context of use. With command help, the user is asking for help at the operating system level. The command takes arguments that specify and refine what it is that the user is asking for help about. Contextual help implies that the system has some information about the user's current state, and can use this information to help display the most relevant information.

With the advances in online assistance and technological advances such as workstations and personal computers, help systems may defy attempts at being placed into any of the categories listed above. In fact, Fenchel's third category may be omitted as a true help system category, insofar as it seems to admit only facilities for collecting user comments.

Sondheimer and Relles [1982] construct a taxonomy by classifying help systems according to the following four categories:

- 1) access method - the way users can construct or enter requests for assistance,

- 2) data structure - the manner in which different portions of assistance information are related to each other,
- 3) software architecture - how assistance requests and their responses are communicated among a user, an operating system, application programs, and the assistance database, and
- 4) contextual knowledge - how much information is retained about the assistance environment, including the user, the application, and the tasks being performed.

Borenstein [1985], asserting that 2, 3, and 4 are primarily implementation considerations, suggests a user-oriented taxonomy of online help across the three general dimensions of

- 1) help access, including access mechanisms,
- 2) help presentation, and
- 3) integration - the degree to which the various help features are uniformly available in all potentially relevant contexts.

Although Borenstein's classification is made from a more user-oriented perspective, he omits from his classification such basic user-oriented help features as error diagnostics, prompting messages, and interface paradigms. These latter features play an important role in the usability and learnability of any system.

Bergman et al. [1985] extend Sondheimer and Relles' classification to include user-interface related features, thereby providing a workable classification scheme for a variety of technologies and systems. Their classification includes the following three extensions:

- 1) add-on versus integrated help - whether the help information is an integral part of the interface or whether the help is offered in addition to and instead of the information normally provided by the interface,
- 2) solicited versus unsolicited help - whether help is provided only as the result of an explicit request for help by the user, or whether the help system uses its own criteria to provide help, even in the absence of a request, and
- 3) specified versus unspecified help - whether the system requires the user to specify what information is needed, or whether the user may make a non-specific request.

It is worth repeating that most help systems will probably never be perfectly categorized into any one classification. Rather, they will be hybrids, incorporating features of several. The challenge is to develop a well-balanced, consistent design that incorporates those features which have been shown to be successful.

2.3 Issues in User Interface and Help System Design

Even the best help system would be useless without a well designed interface. In addition, requirements for the help system may in large part determine the user interface. For this reason, basic principles of interface design must be explored.

User interface design is more accurately termed art than science. Often, there is no "right" answer, or no scientific basis to justify one choice over another. However, empirical studies can provide guidance for the designer.

The designers of the Xerox STAR system [Smith et al. 1982], who pioneered user interface mechanisms like icons and windows, advocate consistency and simplicity throughout. Mechanisms should be used in the same way wherever they occur. Different modes of interaction, in which input may mean one thing in one mode and something entirely different in another, are to be avoided.

Some researchers [Yestingsmeier 1984] advocate early user involvement in interface design, maintaining that users can best describe their needs and expectations from a system. Others [Blake 1986] assert that blindly implementing user preferences can be disastrous. In command naming behavior, for example, users chose high frequency, general commands, inconsistent command arguments, and relied too

heavily on their memory [Blake 1986].

Other user interface issues which have relevance in the design of online assistance are: (1) the use of menus [Hodgson and Ruth 1985], (2) designing systems which minimize user errors [Norman 1983], (3) choice of input mechanisms [Card et al. 1978], (4) window layout and design [Gait 1985; Bly and Rosenberg 1986], and (5) "soft-machine" interfaces [Nakatani 1983].

Research has been done on issues in help system design as well [Connally et al. 1985; Houghton 1984]. More often than not, researchers have started from scratch, only to "rediscover" the basic paradigms. Borenstein [1985] warns that the greatest problem affecting builders of help systems is lack of knowledge of what has already been done. Anyone who wishes to implement a help system must study past and present systems to avoid making common mistakes.

Relles and Price [1981] identify several questions that inevitably arise when considering the development and use of online assistance:

- * How many levels of assistance should be available (i.e., novice, casual user, expert, etc.)?
- * Should the user or the system decide when assistance is necessary?
- * What factors should be considered in the textual composition of online assistance?
- * How can online assistance be provided easily and

efficiently by programmers?

The following sections explore recent research which addresses these issues.

2.3.1 How Many Levels of Help Should be Available?

Schneider [1982] asserts that user sophistication with a software system can be characterized by five levels: parrot, novice, intermediate, expert, and master, and that user assistance information should be factored accordingly. However, many widely used help systems offer users the same depth of information, regardless of the user's expertise (IBM CMS* Help [IBM 1983] and VAX/VMS** Help [DEC 1980] are two better known examples). The Unix system offers two dimensions: syntactic help (e.g., if a command is entered with no arguments) and semantic help (via the "man" command), although the same text is presented to users of all levels of expertise. These systems are little more than online reference manuals. It is important to consider the level and amount of information presented to avoid submerging the user in irrelevant information or information buried so deep it is difficult to find. User sophistication can be classified into various learning stages [Schneider 1982] and software can be designed to accommodate

*IBM/CMS is a trademark of International Business Machines Corporation

**VAX/VMS is a trademark of Digital Equipment Corporation

these stages [Mozeico 1982] by providing multiple entry points; each designed to facilitate movement from one stage of learning to the next by selectively presenting help information.

2.3.2 Should the User or the System Decide When Help is Required?

Most systems are equipped to respond to a user's explicit request for help, but one notable exception is the Interlisp DWIM (Do What I Mean) facility [Teitelman 1984]. When Interlisp encounters incorrect input, it invokes DWIM to diagnose and suggest a correction. The user must still approve DWIM's suggestion, but the system has taken the initiative to provide help. Another example of system initiated help is command completion [Walker 1986]. The user types a few characters, and the system displays the full name once it is unique.

The implementers of TNT (Tutor aNd Trainer), however, elected to leave the initiative to the user, rather than second-guessing what the user wanted [Nakatani et al. 1986]. In a series of laboratory experiments, users found such system behavior irritating and distracting.

2.3.3 What Factors Should be Considered in the Textual Composition of Online Assistance?

Borenstein [1985] surprisingly found that the most important determining factor in the "goodness" of a help system is the quality and nature of the texts it presents.

In addition, it is generally agreed that the composition of help texts be separated from the design and implementation of the system, and should preferably be done by a technical writer.

A related issue concerns the reading level of text presented. One study [Roemer and Champanis 1982] found that subjects prefer a fifth grade reading level for tutorials presented in an interactive environment. Other findings [Houghton 1984] bear out that the most sensible approach to designing help text is to use the simplest possible language, avoiding information overload, and anthropomorphization.

2.3.4 How Can Online Assistance be Provided Easily and Efficiently by Programmers?

Providing good assistance, like providing good documentation, is an unpleasant chore for most programmers. Ideally, the person who incorporates and maintains assistance information in a program should be able to do so with minimal effort and disturbance to that program's design [Sondheimer and Relles 1982]. A centralized assistance processor that interprets requests against an assistance database offers a more consistent environment for users and eliminates redundancy for programmers [Relles and Price 1981; Relles et al. 1981].

2.4 Survey of Some Existing Help Systems

This section surveys a cross-section of some experimental and real-life help systems. No attempt is made to be comprehensive, but a sampling is presented here to give an idea of the variety of approaches that have been taken to provide online assistance. Noteworthy features, advantages and disadvantages of these systems are discussed.

2.4.1 BROWSE

BROWSE [Bramwell 1983], an online manual system for the Berkeley Unix System, is a compact system that makes good use of dumb terminals. It combines menus and keyword searching capabilities. Primitive windowing is performed by dividing the screen into two 11-line displays. Keywords are sought by searching a file containing brief descriptions of documents in the database. It is a stand-alone system, and therefore cannot provide any context-dependent help. It is compact, conceptually simple, and relatively easy to extend.

2.4.2 AT&T Unix System Help Facility

The **AT&T Unix System "Help" Facility** [Butler and Kennedy 1985] consists of the five commands "help", "starter", "glossary", "usage", and "locate", each of which can be accessed at the command level, or by progression through a set of menus. Like BROWSE, it provides an interface between the user and system documentation. It contains a facility (helpadm) whereby system administrators can add information and modify the help database. Implementers

rely heavily on input from both experienced and inexperienced Unix users to decide what to include in the help system. The system is primarily menu-driven, but the information is packed so densely that conceptually, the menus are never more than three layers deep.

2.4.3 ICNHELP

ICNHELP (Integrated Computer Network Help) [Stoddard et al. 1985] is designed as a summary, or mini-reference system, for experienced users who know what they are looking for, but need memory refreshers on exact command usage. Access to help is hierarchical by means of menus, and is patterned after the VAX/VMS help utility. As such, it is more interesting for what it is intended to do than for how it does it. Surveys after the first 5 months of use found that merely making an online help system available does not necessarily change the patterns of users already satisfied with printed documentation.

2.4.4 SIGMA

The help facility for the **SIGMA** message service [Rothenberg 1979] consists of online access to reference materials and a Tutor which provides lessons and exercises. If commands are erroneously entered, the system guides the user to help correct the error. If more detailed description is needed, the user can request the next level of help. It incorporates many features that were rarely found in early help systems, like context-sensitive help, a

spelling correction algorithm, Flash and Status windows, and tutorial exercises in a "protected" environment. A weakness is that the same text base is used for composition of both the written and online information. It seems the major concern of the implementors in this respect is how to present hardcopy text in a manner suitable for online display.

2.4.5 ZOG

The **ZOG** system [Robertson 1981] is not a help system in the sense defined in this thesis, but an approach to man-machine communication. A ZOG system is based entirely on menu navigation, as is its help system. The main disadvantage is its size and complexity: in real-life applications, building a ZOG "net", a network of highly integrated menus and scripts, can be impractical.

2.4.6 TNT

TNT (Tutor aNd Trainer) [Nakatani 1986] represents an attempt to break from traditional online tutoring techniques by using a voice simulator to do nearly all prompting. Its main disadvantage is a lack of "unobtrusiveness" as it often offers help and prompts before a user is ready.

2.2.7 SARA

SARA (System ARchitect's Apprentice) [Estrin et al. 1986] is an integral help system which keeps the state of the user's interaction to provide context-sensitive help. By making use of a context-free grammar and the current parse

state, the assistance processor gives timely and specific diagnostic help.

2.4.8 HELP: A Question Answering System

HELP: A Question Answering System [Roberts 1970] is one of the first help facilities to allow unconstructed natural language input. Keywords are extracted from a request and matched against designated portions of a documentation file.

2.4.9 ACRONYM

ACRONYM [Borenstein 1985], yet another help facility for the Unix system, is a research prototype intended not to break new ground, but to consolidate and integrate existing techniques. A single assistance database can be accessed via a number of independent mechanisms. Emphasis is placed on quality help texts. The author claims that the system would be improved by including tutorials, multiple levels of explanation, special information for experts, customization of information and references to external sources of help.

2.4.10 DOMAIN/DELPHI

DOMAIN/DELPHI [Orwick et al. 1986] retrieves and displays documentation in a networked workstation environment. DELPHI incorporates a graphical, menu-driven user interface, and operates on a library metaphor. Users can browse through a table of contents or conduct a search for a document by topic. One of the most important considerations in the

design was extensibility. A limitation of the system is that there is no context-sensitive help, although this is included in future research objectives.

2.4.11 UC: The Unix Consultant

UC: The Unix Consultant [Chin 1986] is a natural language computer system that advises the user in the Unix operating system. The user can ask UC to produce plans for doing things in Unix, obtain online definitions of Unix terminology, and get help debugging problems. The system, while slow, demonstrates that natural language may eventually be practical from an implementation standpoint. However, results reported in [Borenstein 1985] cast doubt on the usefulness of natural language in a help system.

3.0 DESIGN AND IMPLEMENTATION OF THE PROTOTYPE

This chapter studies in detail the design and implementation of the Assistance Manager for the SMDE. The first part of the chapter discusses factors in the design of the Assistance Manager and the goal which guided the design. Section 3.2 gives an overview of the hardware and software environment in which the Assistance Manager is developed and where it is used. Section 3.3 describes the components which comprise the Assistance Manager package, and Section 3.4 summarizes details of the implementation.

3.1 Motivation: Factors in the Design of the Assistance Manager

Before embarking on the design of the Assistance Manager, an in-depth literature review serves to identify relevant issues in the design of online assistance. Questions such as "Why is online assistance necessary for the SMDE?", "What will the Assistance Manager be expected to do?", and "What design strategy will work best?", need to be addressed. The results of this literature review are reported in Chapter 2.

The research survey also identifies characteristics considered important for successful online help systems. Section 3.1.1 explores these characteristics. Section 3.1.2 discusses the design requirements, and Section 3.1.3 explains our choice of prototyping as a design strategy.

3.1.1 Characteristics of Effective Online Assistance

A review of the literature has helped to identify characteristics that are associated with successful assistance systems [Relles et al. 1981; Smith et al.; Walker 1986; Sondheimer 1982]. The characteristics are listed alphabetically below:

Completeness of textual material is required if the user is to have faith in the assistance system. Incomplete help can be as bad as no help at all, leaving the user frustrated and confused as to what action to take.

Consistency is important both in accessing help and in the conventions used to present help text, prompts, and diagnostics. Assistance should be requested in a similar manner in all of the interactive programs that make up a larger system.

Context sensitivity refers to the system's ability to provide help relevant to the current situation. This implies that the system is keeping track of the user's current state of interaction. The main importance of context sensitive help is that the user does not have to deal with information that is irrelevant or inapplicable.

Expandability enables future growth for an open-ended environment. Related to expandability is maintainability. It is a must that the system be easily updated to keep the help texts current and complete.

Flexibility implies that different levels of user expertise are accounted for in the presentation, composition, and access of help information. The system should be terse enough for more experienced users, yet simple enough for beginners to quickly become confident in their use of the system.

Understandability is achieved by use of clear, concise language, use of technical terms only when necessary, and the judicious use of such display features as highlighting and white space to demarcate related pieces of text.

Unobtrusiveness refers to the ability to request assistance without interrupting the task at hand. Too often, the user must explicitly save his state and then restore it after help is obtained. Unobtrusiveness also deals with the way help is accessed. Help should be immediately available, but not noticed until needed.

We have included the characteristics above in the requirements for the SMDE Assistance Manager. Many are "common sense" in nature, and it is intuitively obvious why these characteristics would be considered desirable. In spite of this, few if any help systems demonstrate all of these traits in more than a cursory way. The reasons for the lack of commitment vary from a failure to recognize the importance of online assistance, to insufficient resources for their design and implementation. By recognizing their importance early in the design phase, these features can be

used to guide design decisions.

The most direct benefit derived from these characteristics is in the encouragement the user feels for using the help system and making progress in the task at hand.

3.1.2 Assistance Manager Design Objectives and Requirements

The Assistance Manager is one of the basic tools required for the development and execution of a model in a Simulation Model Development Environment. Its fundamental requirements as outlined in [Balci 1986], are to

- 1) administer the assistance database,
- 2) provide information on how to use an SMDE tool
or CLI (command language interpreter) command,
- 3) provide the definition of a technical term encountered
in documentation and communication, and
- 4) provide tutorial assistance as appropriate.

This section expands and elaborates on these fundamental requirements.

3.1.2.1 Design Objectives

The overall goal of the Assistance Manager is to provide effective and efficient transfer of assistance information to a user of the SMDE. "Effective" means accurate information is provided which is relevant to the user's needs. "Efficient" implies that if the user is involved in interaction with the SMDE, the switching of

tasks or modes in the process of seeking help is unnecessary. The objectives identified below are intended to meet the overall goal of providing effective and efficient transfer of assistance information. To meet this goal, the Assistance Manager is required to:

- 1) Provide general information for beginning system users. Such information would serve to acquaint new users with the environment, and establish a context for subsequent learning.
- 2) Provide detailed and specific help on the use of an SMDE tool or CLI command.
- 3) Provide for definitions and example usage of technical terms encountered in documentation and communication within the environment.
- 4) Provide tutorial assistance for users of the environment. The tutorial should give the user a protected arena for limited experimentation with a tool's features.
- 5) Provide help that is constantly available and immediately accessible. There should be methods to temporarily suspend interaction with the Assistance Manager, or save the current display for future reference. The user should not be required to step through an artificial protocol or syntax to access immediate assistance.
- 6) Provide help that is unobtrusive; i.e., messages or

prompts that are only visible when required or asked for.

- 7) Provide a help system that is flexible enough to accommodate experience users as well as novice or casual users.
- 8) Provide context-sensitive help wherever possible. The system should use all available information on the user's state and avoid placing the burden on the user.
- 9) Provide appropriate methods of access to the help information. Initiative, mechanisms, and complexity of access should vary according to task and type of user.
- 10) Provide a straightforward and systematic method for tool developers (application programmers) to build help into tools which may be added to the environment.
- 11) Administer the assistance database by serving as an interface between the user or programmer and the database contents.
- 12) Provide help that is available in a consistent manner from any tool within the environment.
- 13) Be easily maintainable and expandable. This is critical in order to accommodate the tailoring and updates that are inevitable in a large software environment. Updates should be enforced in a

manner which helps enforce database integrity and consistency.

3.1.2.2 Functional Requirements

The functional requirements of the Assistance Manager are broken down into two categories: requirements from the modeler's point of view (user interface) and requirements from the point of view of a developer wishing to include help in a software tool (programmer interface). A modeler within the SMDE has access to four functional components of the Assistance Manager: Introduction to SMDE, Glossary, Tutorial, and Local Help.

The Introduction to the SMDE gives beginning information for new users, providing a context for subsequent learning. It lists the components of the environment and their features. It differs from a tutorial in that it tells "what", not "how".

The Glossary gives technical definitions and definitions of terms specific to the environment.

The Tutorial gives detailed instructions on the tool's use and may be used for limited experimentation with the tool.

The Local Help is available through any SMDE tool. It explains a tool's features and provides assistance for recovery in the event of user error.

The Assistance Manager requires a programmer interface to enable tool developers to include help with their code.

The second category of functional requirements, Programmer Assistance, is necessary for adding and updating help scripts, tutorials, documents and definitions to the assistance database.

3.1.3 Prototyping as a Design Strategy

A software prototype is a functionally incomplete model of a proposed system, built to demonstrate feasibility or explore potential requirements [Church et al. 1986]. Prototyping has been used most frequently to gain an understanding of user requirements, and has recently become a technique of interest in designing software systems. The prototype may be built quickly, placed into use, and studied to see where design flaws are evident.

Typically, the prototype lacks a full complement of functions. Concerns of efficiency and elegance take second place to the concern of bringing up the system quickly.

Prototyping is the design technique of choice for the entire tool set of the SMDE, and in particular, the Assistance Manager. By building the Assistance Manager quickly, we create a testbed to evaluate such features as its functional components, user interface, and its methods for help development and integration by application programmers. This proves extremely useful in an environment, such as ours, where we have yet to grasp the limits of the technology we are using in development. It

may be impossible in some instances to determine the feasibility of a design decision until an attempt is made to implement that feature with our hardware/software configuration. If the designer has insufficient knowledge of the system's capabilities, then it is difficult to specify features that can actually be implemented. In the case of the Assistance Manager, the solution of the design problems went hand in hand with the analysis of the hardware/software capabilities.

3.2 Hardware/Software Environment

Before describing the prototype of the Assistance Manager, several essential aspects of the hardware and software configuration should be pointed out. The architecture of choice for the prototyping of the SMDE, and in particular the Assistance Manager, was a SUN 2/160UC* color workstation running a Berkeley 4.2 based Unix operating system. The SUN is a 2 MIPS machine consisting of a 16.67-MHz MC68020 microprocessor with a 16.67-MHz MC68881 floating-point coprocessor, a 380MB Fujitsu Eagle disk subsystem, a 1/4-inch cartridge tape subsystem, 4MB of main memory, a 19-inch color monitor with a resolution of 1152 X 900 pixels, a

*SUN 2/160 is a trademark of Sun Microsystems, Inc.

pointing device called a mouse, and a connection to the Ethernet network which enables high speed file transfer to and from other University computing systems.

The most important ingredient of the user interface is the 19-inch bit-mapped display screen. Because every screen pixel can be turned on and off, the SUN has an excellent ability to present visual images. Much of the work done for this thesis deals with this feature of the system; i.e., the user interface capabilities.

The user communicates with the SUN by keyboard input. In addition, he may use the mouse to point to or select locations on the screen. The system provides continuous feedback as to where the mouse is pointing by displaying a cursor on the screen. Virtually any material that is displayed on the screen can be pointed at and treated as input.

The user views the environment through a display consisting of rectangular "windows". Windows are self-contained work areas and are analogous to sheets of paper on a desk top. They can be overlapped on the screen, effectively increasing the user's working space.

Windows may be resized, "closed" or collapsed to a small figure commonly called an icon, and subsequently "reopened". Within a window, text may be scrolled for viewing.

Each window corresponds to a different task or aspect of the user's environment. A user can switch back and forth between tasks, which is of great value when the tasks are related in some way. Thus, a user may be editing a file on one window, running a program in another, checking a file listing in yet another, etc. The SUN's window management system takes care of updating the display image, tracking user input, and similar concerns.

One technique heavily used throughout the system is the use of menus. Menus facilitate context switching between tasks because the user does not have to remember a command. Rather, he may use the mouse to select one of a finite set of options on a menu. Often, the menus are "pop-up", appearing only as a result of pressing a button on the mouse. This allows for conservation of screen space and a high degree of unobtrusiveness. Menu options are frequently selected by a software "button", a button-like image over which the user can position the mouse cursor and "select" by pressing a button on the mouse.

Without the hardware and software configuration just described, it would have been impossible to design an Assistance Manager which could hope to embody the desirable characteristics described in Section 3.1.1.

3.3 Components of the Assistance Manager

This section illustrates in detail the components

designed to meet the functional requirements of the SMDE Assistance Manager which were outlined in Section 3.1.2.2.

The Assistance Manager may be run as an independent tool by invoking it through the Unix shell (Fig. 3.3.1), or it may be invoked by selecting from a pop-up menu available on any other SMDE tool (Fig. 3.3.2). In the latter case, the Assistance Manager is spawned as a separate process. By enabling access to the Assistance Manager at the Unix command level, a user may learn about the SMDE without actually entering it.

Once the Assistance Manager has been invoked, a user may use the mouse to select any of its components. Because each component is a process in itself, it is possible to run any number of them simultaneously. When not in use, the component may be collapsed into an icon to conserve screen space. The architecture of the Assistance Manager is shown in Appendix 2.

3.3.1 Introduction to the SMDE

The Introduction to the SMDE is intended to provide general information for beginning users of the environment. It discusses SMDE tool features and uses. In this sense, it may be considered as the online documentation component of the Assistance Manager, although it makes no attempt to serve as a reference manual. The Introduction gives references for the curious user who wishes to find out more about the concepts and methodology that underlie the SMDE.

Fig. 3.3.1 Assistance Manager Tool Menu

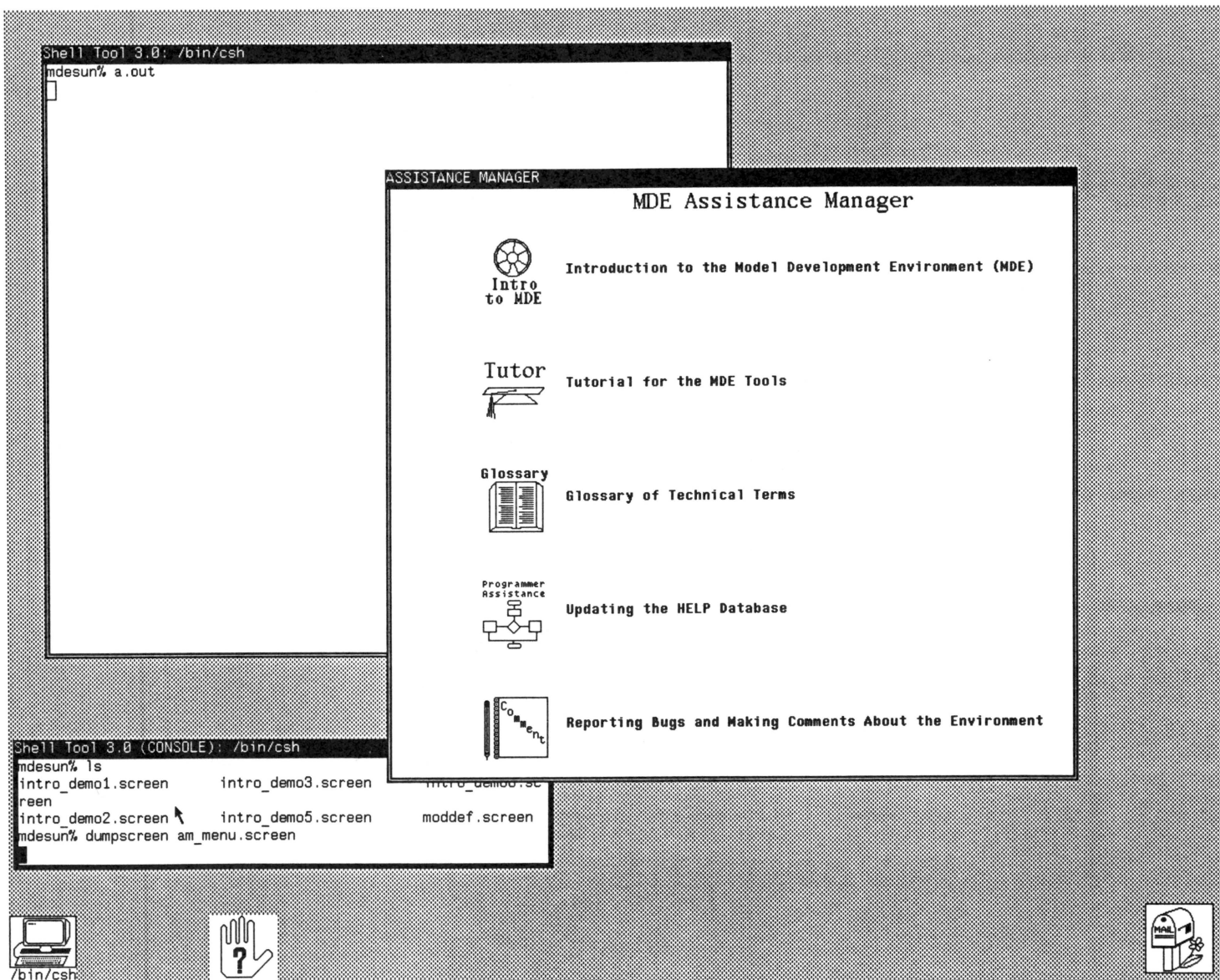
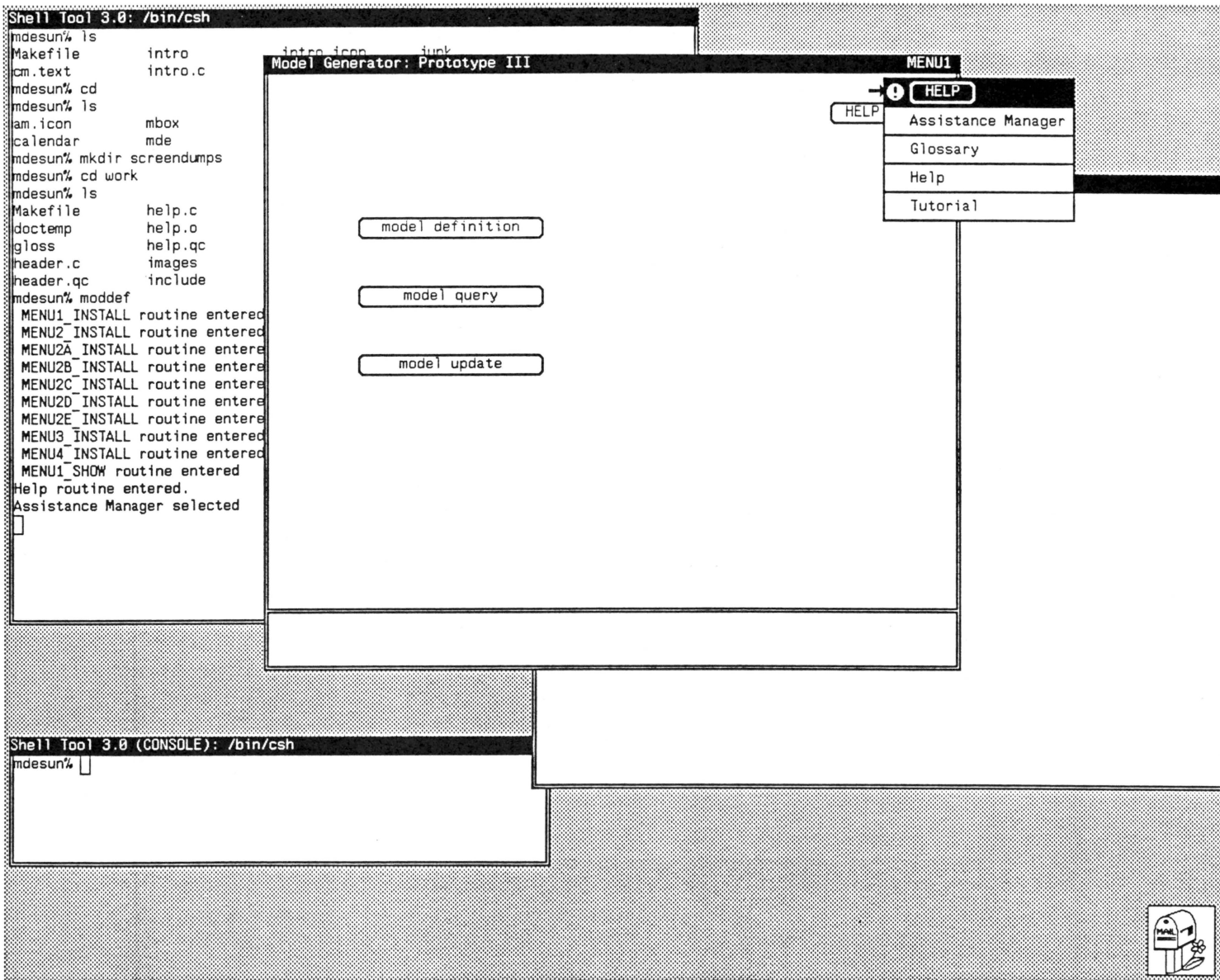


Fig. 3.3.2 Invoking the Assistance Manager through an SMDF Tool



The interface to the Introduction consists of two viewing windows and one control panel (Fig. 3.3.1.1). One viewing window is provided for text, and another is provided for the table of contents of the text being viewed. Both viewing windows may be scrolled up or down by using the mouse, and either window may be further split into multiple views of the same text.

The control panel allows the user to select a topic from the table of contents for display, or to search for a particular topic (Fig. 3.3.1.2). A "Show Documents" button is provided to return the user to the top level table of contents, which lists all chapters available for viewing. An exit button is provided to leave the Introduction. (The user can also exit, if he wishes, by using the standard pop-up menu on the tool border stripe.)

The Assistance Manager provides the mechanism by which documents are updated or added to the documents database. The document writer develops and formats the text for display using a text editor and/or formatter. Once the text is in a displayable format, the writer adds the document to the documents database by accessing the Assistance Manager and selecting "Programmer Assistance". This feature will be further explored in Section 3.3.6.

3.3.2 Tutorial

The Tutorial component, like the Introduction to the

Fig. 3.3.1.1 The Introduction to the SMDE Component of the Assistance Manager

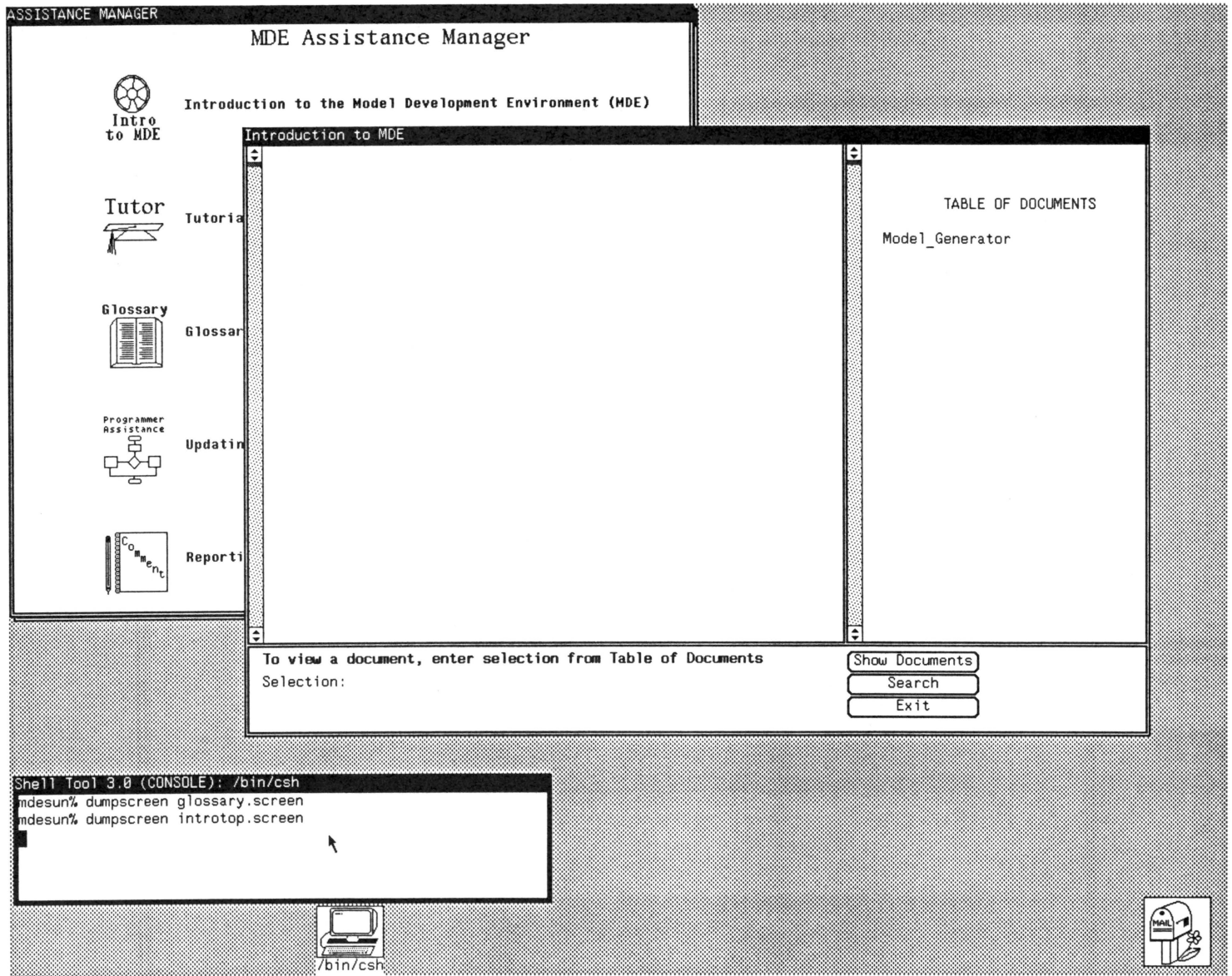
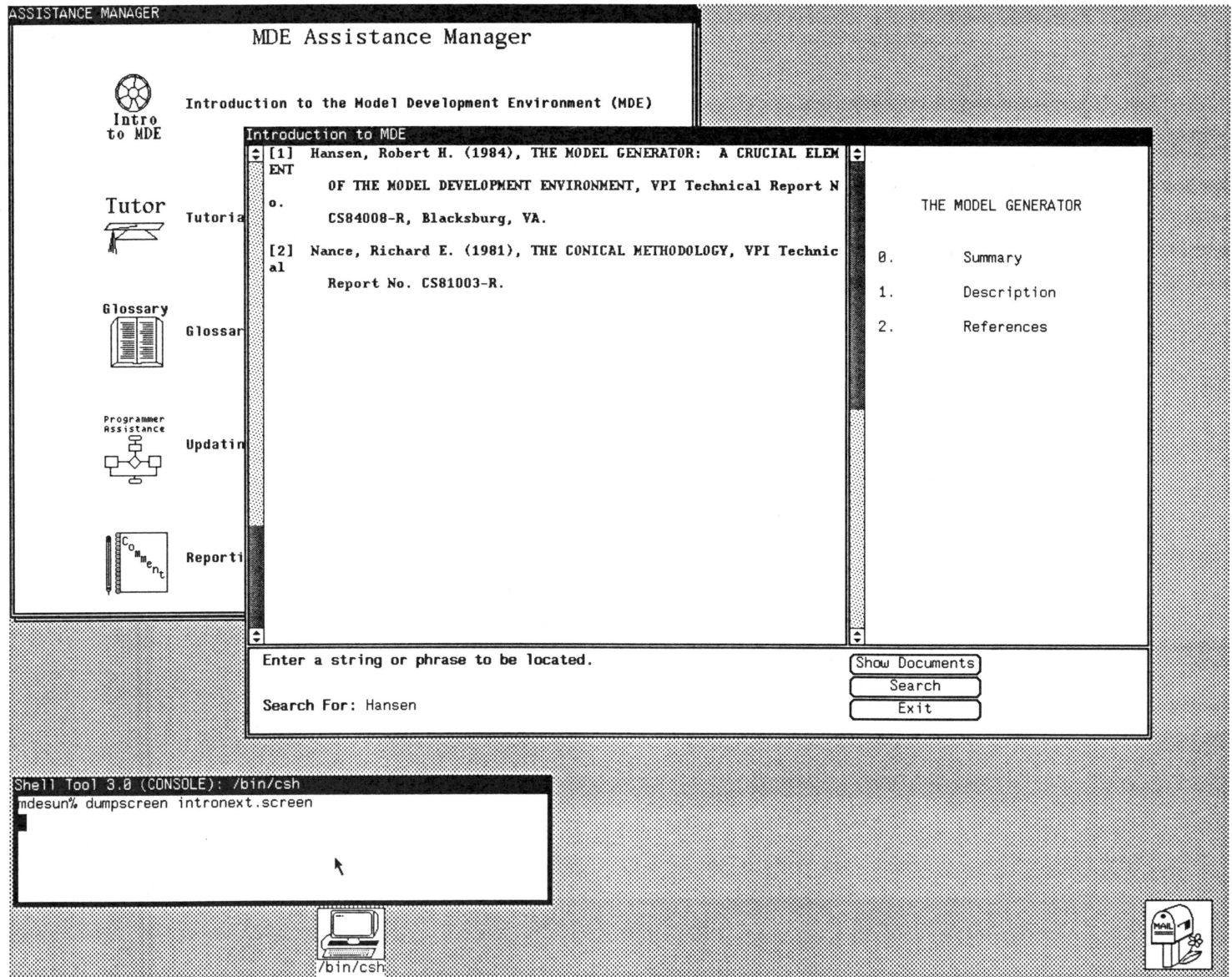


Fig. 3.3.1.2 Searching for a Phrase Within the Introduction to the SMDE



SMDE component, is made up of an interface and a database of documents. The documents, however, will differ considerably from the type used in the Introduction. Whereas the Introduction tells what the SMDE is all about, the Tutorial deals with how to use the SMDE tools.

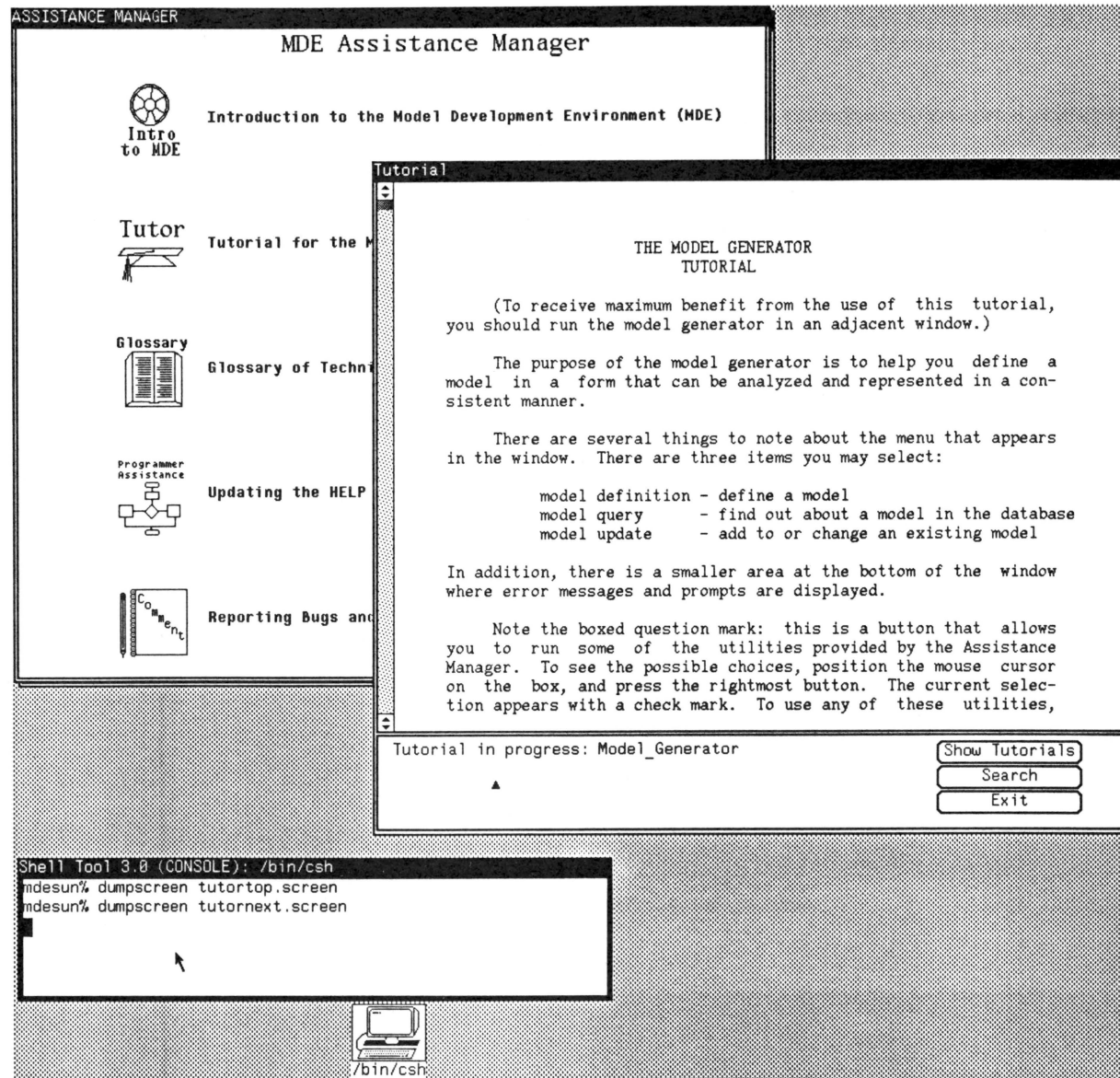
The interface to the Tutorial consists of a viewing window and a control panel (Fig. 3.3.2.1). The viewing window displays either a list of available tutorials or the text of a tutorial which the user has selected. The text in the viewing window may be scrolled up or down using the mouse, and may be further split to allow multiple views of the text.

The control panel consists of a status prompt and buttons which allow the user to list all available tutorials, search for a string, phrase, or topic in the text, or exit from the Tutorial. The Tutorial runs as a separate process, and may be invoked either from the top level Assistance Manager menu or from an SMDE tool. When not in use, it may be exited or closed to an icon to conserve screen space.

Ideally, the Tutorial is run in a window alongside the tool of interest, giving the user step-by-step instructions on how to proceed. This approach gives the interactive behavior necessary for effective tutoring, yet removes the burden of interaction from the Tutorial itself.

The Tutorial for an SMDE tool should be created by the

Fig. 3.3.2.1 The Tutorial Component of the Assistance Manager



tool developer or someone else who is knowledgeable in the use of the tool. The text of the tutorial is prepared with a text editor and/or formatter. Once the tutorial is in a suitable format, it is added to the tutorial database in a manner similar to that used for the Introduction documents.

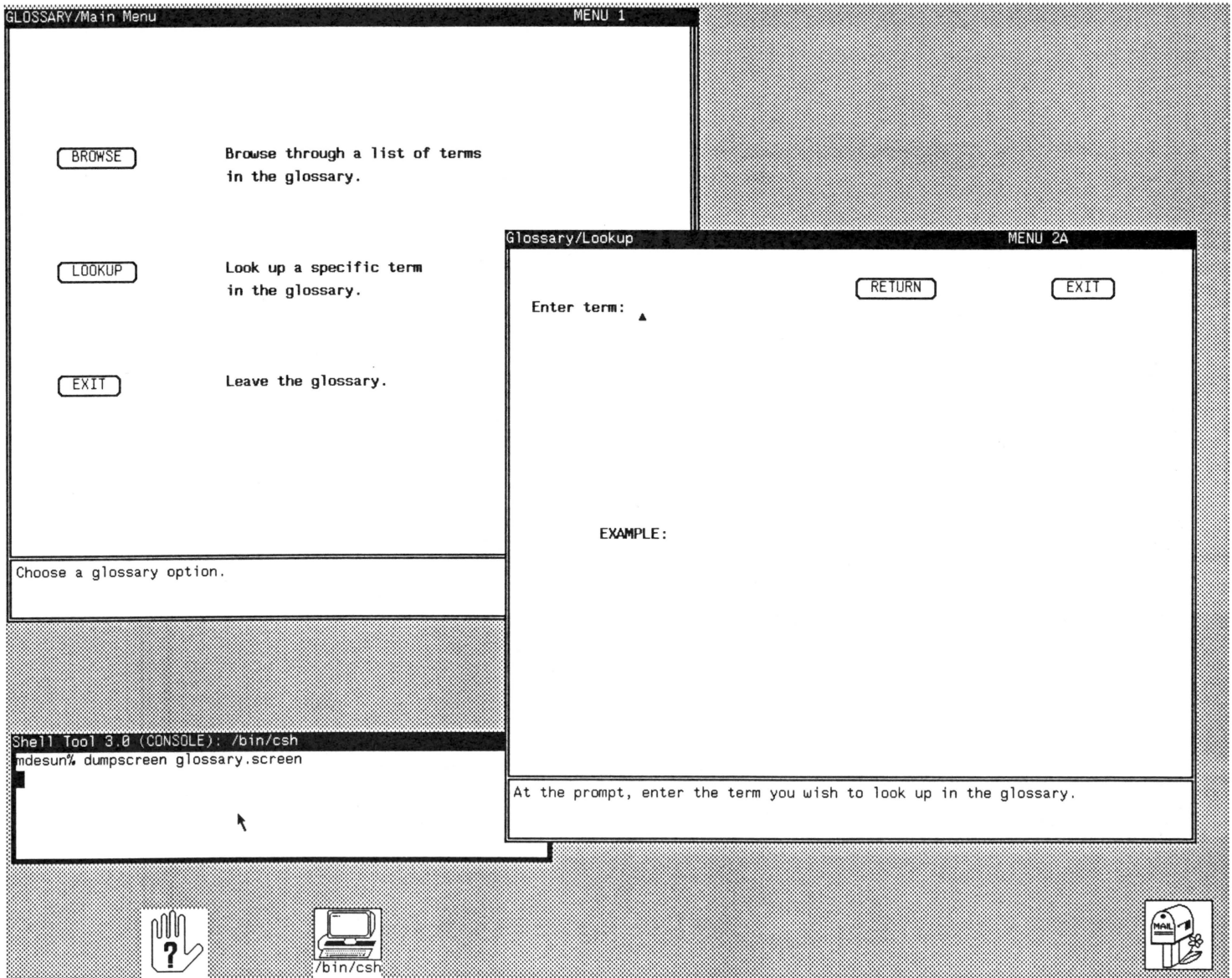
3.3.3 Glossary

The Assistance Manager provides a glossary feature which can be used to find the meaning of a technical term or a term which has a definition specific to an application. The glossary runs in its own window and interfaces directly to the glossary database. Like the other functions which can be chosen from the Assistance Manager menu, the glossary may also be invoked from a tool within the SMDE and runs independently of the calling process. When not in use, it may be exited or collapsed to an icon to conserve screen space.

As shown in Fig. 3.3.3.1, a user calling the glossary is confronted by a window containing a status area and a menu of possible activities in the glossary. The user selects an option using the mouse.

The lookup option provides direct access to a term in the glossary. If the user selects lookup, the menu is replaced by a new menu (Fig. 3.3.3.1) which first prompts for the term in question, and then displays its definition and an example of its usage or meaning. If the term is not

Fig. 3.3.3.1 Glossary Main Menu (bottom left) and Glossary Lookup Menu (upper right)



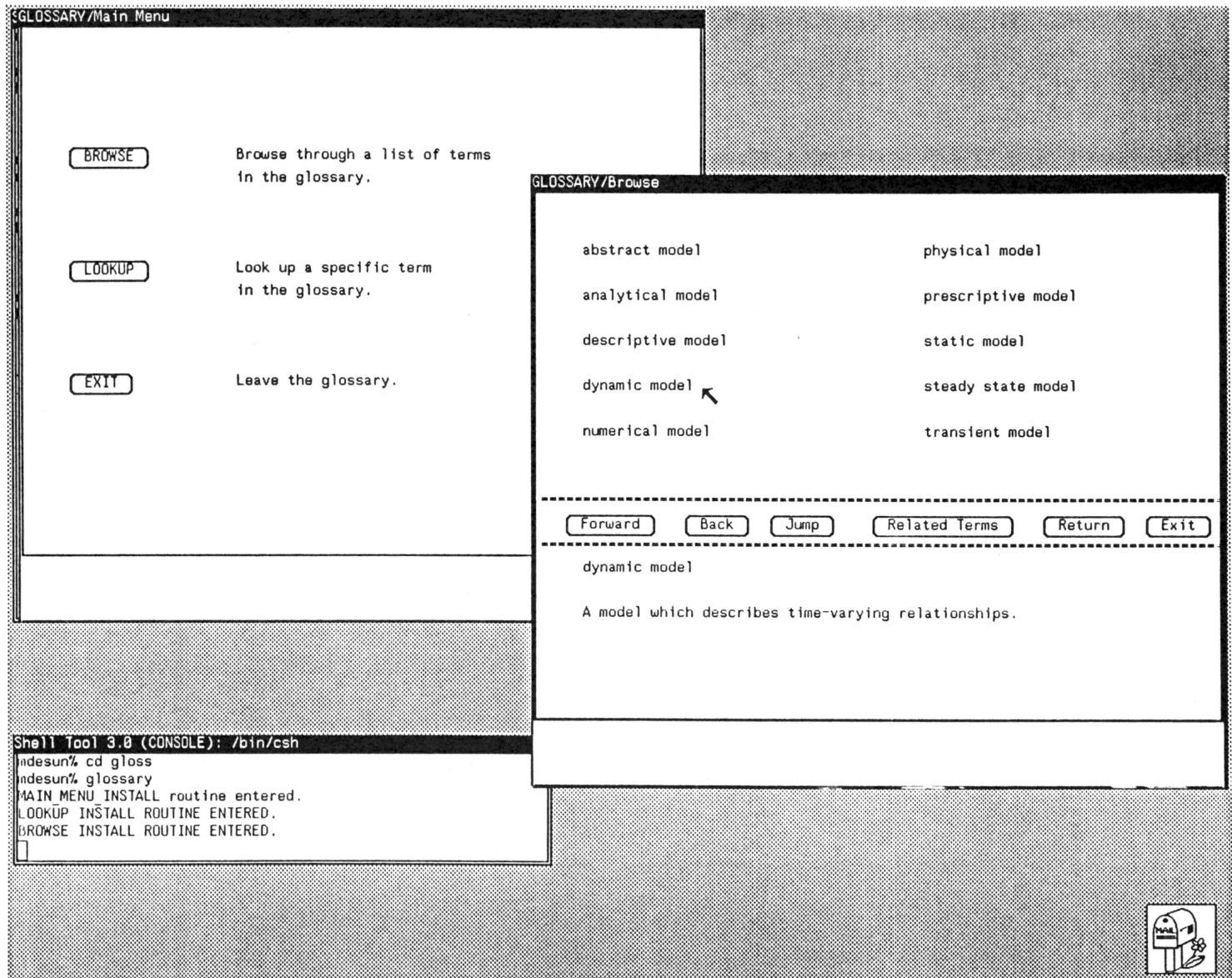
contained in the glossary, a message to that effect is displayed. The status area instructs the user on window use. At this level, the user may return to the main menu, exit the glossary, or collapse the glossary into an icon.

The user who wishes to browse through the glossary may select the browse option from the glossary main menu. A screen appears which is considerably more complex, in keeping with the more advanced functionality provided (Fig. 3.3.3.2). This window contains a display area, a control panel, a viewing area, and a status area.

By default, the first ten terms in the glossary appear immediately in the display area. The user selects "Forward" or "Backward" to view additional terms. To display a term's definition, the user positions the mouse cursor over the term and selects by depressing the first button on the mouse. The term is highlighted in reverse video and the definition is displayed in the viewing area. Subsequent selections from the display area cause the previous definition to be overlaid by newly selected ones.

If the user wishes to examine related terms, the control panel button labeled "Related Terms" may be selected. The Glossary prompts the user to enter a term for which related information is sought. If any related terms exist in the database, these will appear in the display area and may then be selected for definition.

Fig. 3.3.3.2 Glossary Browse Menu



By choosing the "Jump" option, the user may be positioned at an arbitrary location in the Glossary. This is a useful feature for moving around quickly when the Glossary contains many terms. The user is prompted to enter a term or letter of the alphabet. Once provided, the terms are listed beginning at this alphabetical location.

The status window keeps track of the user's interaction and provides terse instructional messages. At any time, the user may return to the Glossary main menu, collapse the Glossary into its icon representation, or exit.

The Glossary contents are updated and expanded through the Programmer's Assistance component of the Assistance Manager.

3.3.4 Local (Tool-Specific) Help

Local Help refers to help available within a tool running in the SMDE. Such help is accessible in a uniform, consistent manner and meet the requirements of flexibility, expandability, unobtrusiveness, context-sensitivity, consistency, and maintainability. The Assistance Manager provides the means for this by giving tool programmers the ability to include a help package directly in their code.

Once inside an SMDE tool, the user may access help in any of three ways. Via a button selection from the tool window, the user may choose from a pop-up menu to go to the Introduction to the SMDE, Tutorial, or Glossary (Fig.

3.3.2). Once the selection has been made, a new window appears and the Assistance Manager component runs in it independently of the tool which invoked it.

A user's interaction with a tool is typically driven by panel items which allow the user to select options and enter text. Help for these functions is enabled by positioning the mouse cursor over the panel item and depressing the middle mouse button. A small pop-up window appears inside the tool window with an explanatory or instructional message (Fig. 3.3.4.1). The window remains visible until the user selects "Done" from the help frame, at which time it disappears.

Each tool window contains a small status area where error diagnostics and messages appear. A "Help-on-Tap" feature is provided via an "Explain" button which is displayed when an error message or prompt occurs. Selecting the "Explain" button results in detailed instructions or explanations of error conditions and why they may have occurred. The button may be selected repeatedly for progressively more detailed information (Fig. 3.3.4.2). The button does not appear on the screen unless there is text displayed in the status window.

The features just described are achieved through a comprehensive set of modules which the tool programmer includes in his application code. Once the package is included, the programmer needs to make specific information

Fig. 3.3.4.1 Example Demonstrating Help Frame Feature of Local Help

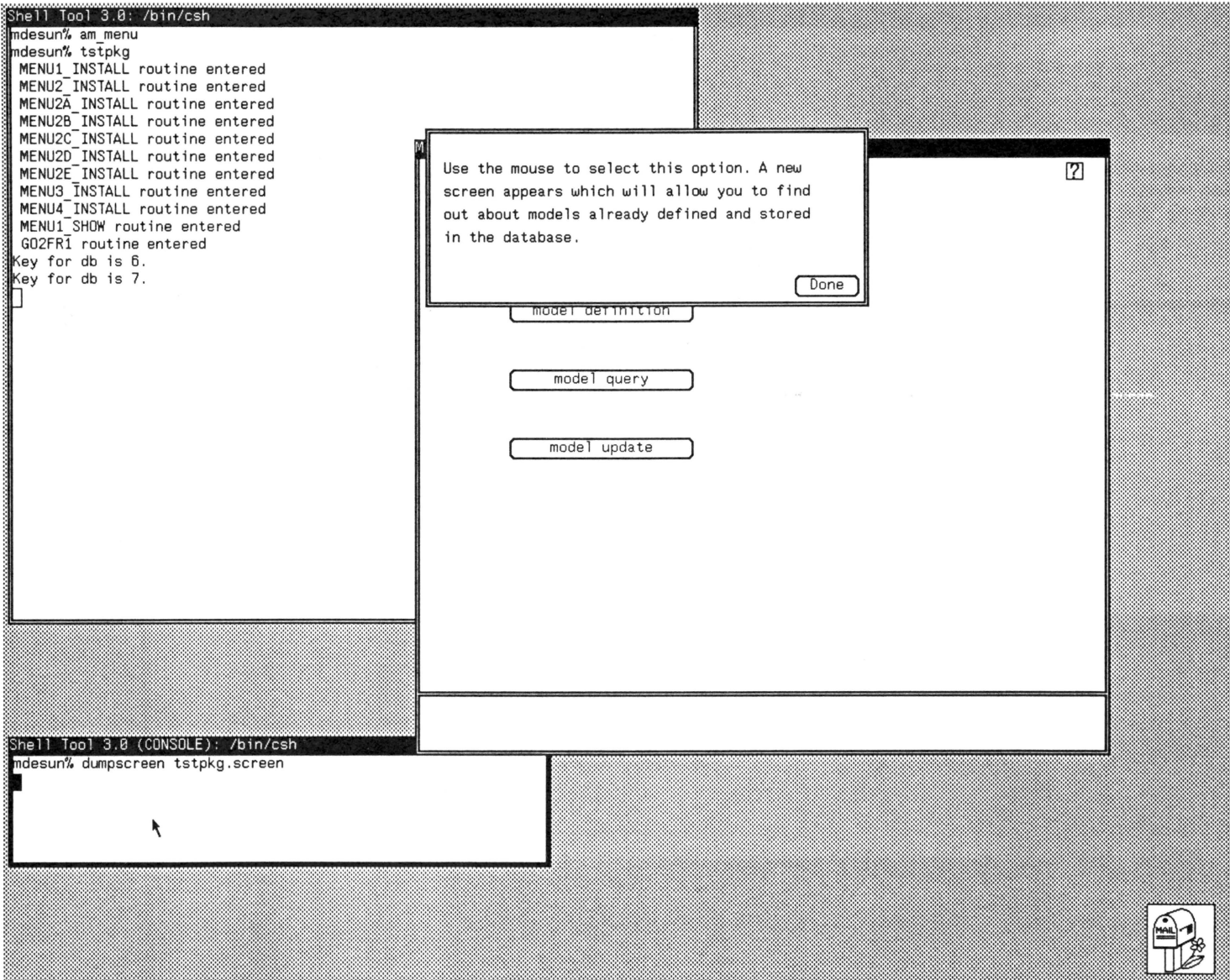
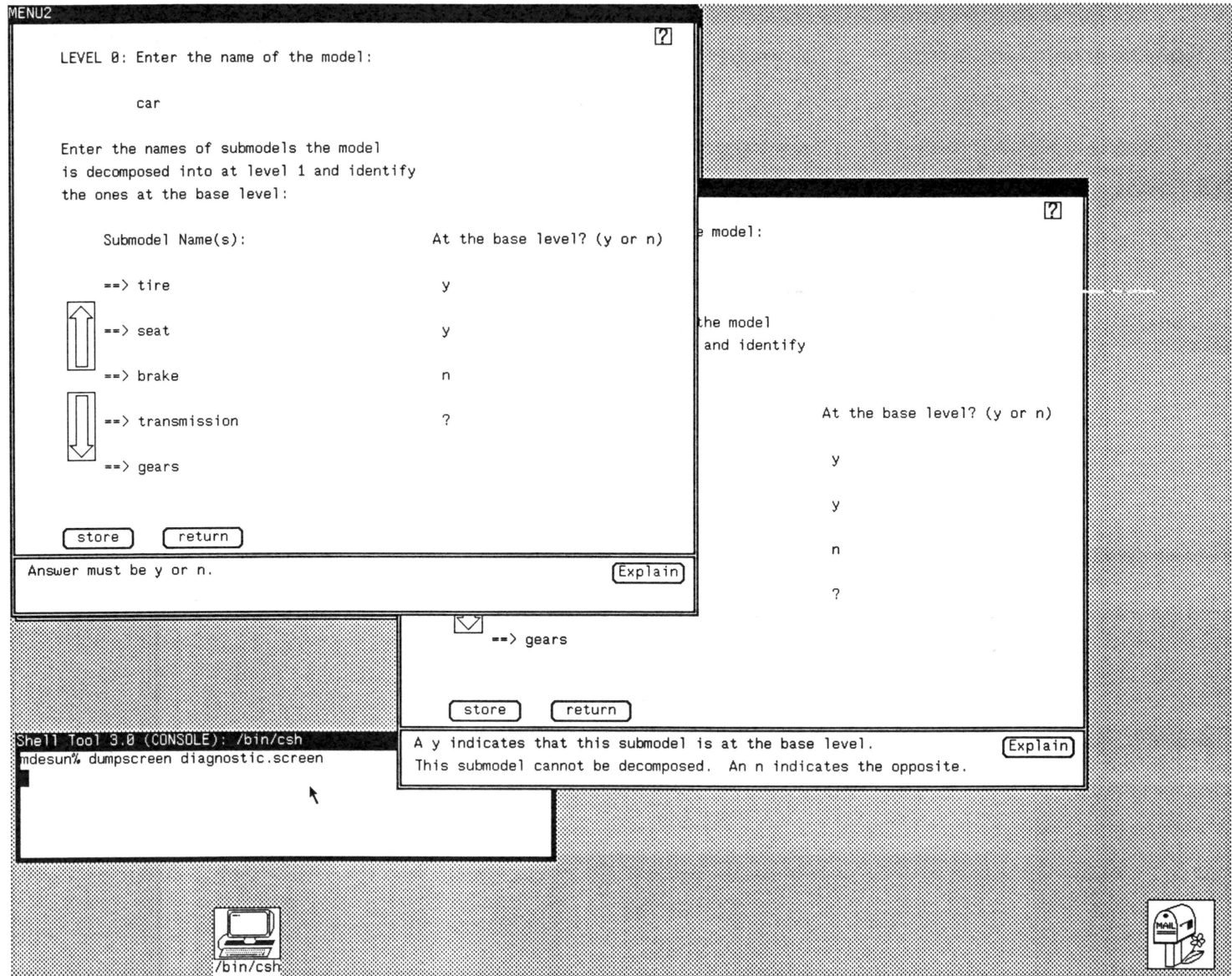


Fig. 3.3.4.2 Example Demonstrating the "Explain" Feature of Local Help



available to the help package. Although this results in extra code for the application programmer, this code is nevertheless straightforward and easy to implement.

The text for the help messages and error explanations will be prepared independently of the application program, and must be stored in the help database. Apart from the composition of the help scripts, this task will be automated and is further described in Section 3.3.6.

3.3.5 Comment Facility

A comment option is included in the prototype to record user observations and suggestions (Fig. 3.3.5.1). The comments are stored in the help database along with the user's name, date, and the tool in use at the time the comment was made. This facility will provide a means to track user satisfaction with the Assistance Manager, and will also be useful for reporting "bugs" in the design or implementation.


3.3.6 Programmer Assistance

Programmer Assistance deals with updating and maintaining the help database, and the use of the help package in an application program. Unlike the other components of the Assistance Manager, Programmer Assistance is available only to developers who have access authorization for its use, and is only available through the top-level Assistance Manager menu (Fig. 3.3.6.1).

Fig. 3.3.5.1 Comment Form (top right)


ASSISTANCE MANAGER

MDE Assistance Manager




Intro to MDE

Introduction to the Model Development Environment (MDE)




Tutor

Tutorial for the MDE Tools




Glossary

Glossary of Technical Terms



Programmer Assistance

Updating the HELP Database



Comment

Reporting Bugs and Making Comments About

Assistance Manager Comment Form

MDE COMMENT FORM

Enter your comments on the form below. Use the <TAB> key to move between fields.

Name: Date:

Tool:

Comment:

IF YOU NEED MORE ROOM, USE ANOTHER COMMENT FORM.

Press <ESC> and type "APPEND" to add, "EXIT" to quit.

Help Append(control_F) Exit

Shell Tool 3.0 (CONSOLE): /bin/csh

mdesun% dumscreen comment.screen


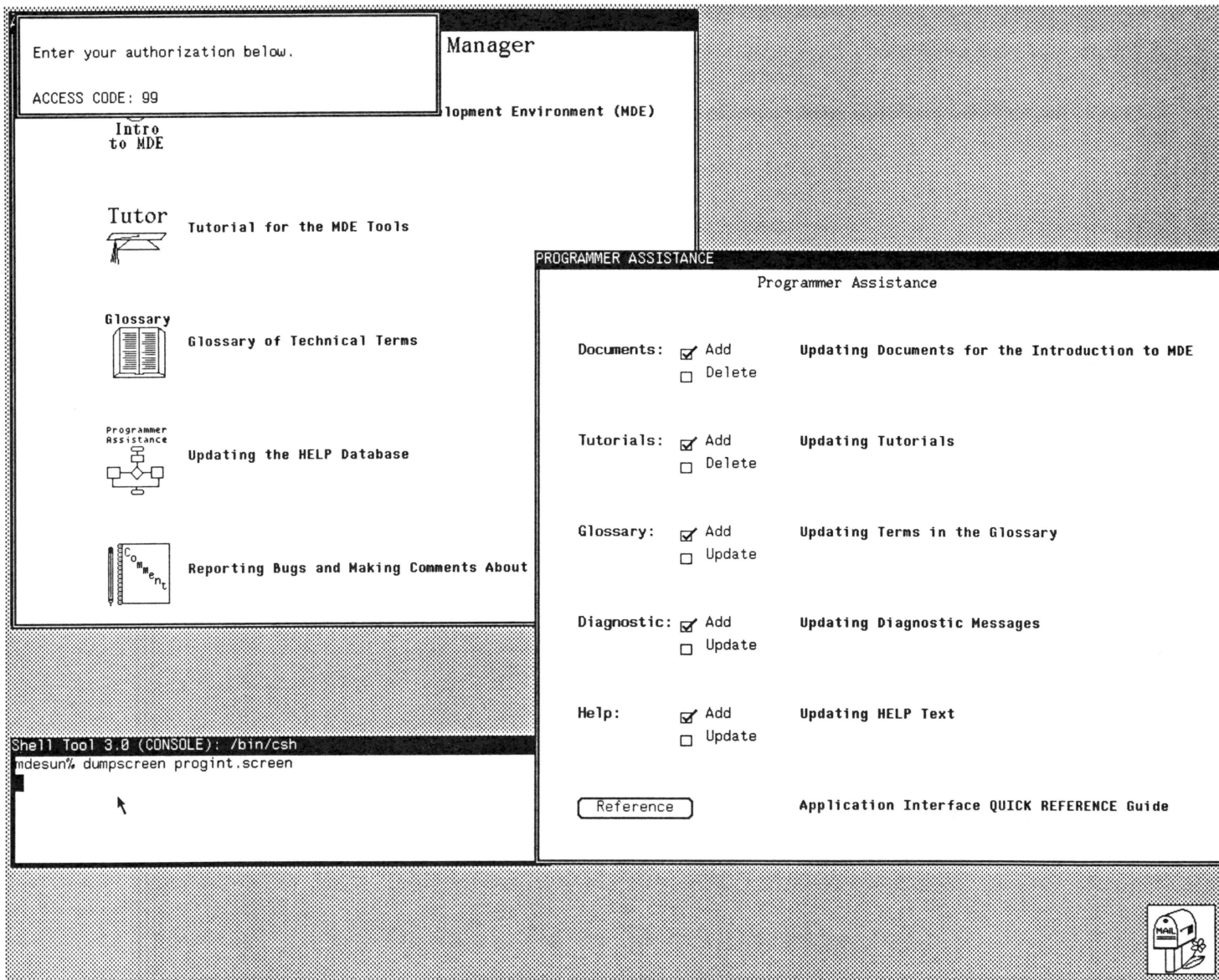


Fig. 3.3.6.1 Programmer Assistance Main Menu (top right)



When a user selects Programmer Assistance from the Assistance Manager menu, a pop-up frame appears and prompts for an access code. If an incorrect code is entered, the tool denies access and the frame disappears. If the code is a valid authorization code, the pop-up frame disappears, and a new tool window appears which lists the options available under Programmer Assistance (Fig. 3.3.6.1).

The options allow the user to add or delete from the tutorials and documents databases, and to add, delete, and update the glossary definitions, diagnostic messages, and help scripts in the help database.

Adding and deleting tutorials for the Tutorial and documents for the Introduction to the SMDE are handled similarly. Once the user selects the specific option, the Programmer Assistance window is overlaid by a new window which prompts the user for document name and location (Fig. 3.3.6.2). The documents are moved to the appropriate directory and the Table of Contents is updated.

Updating the glossary, diagnostic, and help script databases is accomplished by a direct interface to the database. The programmer may retrieve text for update or add new material at any time, although in most applications this would be done only after the tool developer had completed testing and debugging of his design. Invoking either the add or update option results in the replacement of the current screen by a new screen which is tailored for

Fig. 3.3.6.2 Example of Interaction to Add Documents to the Documents Directory

```
Shell Tool 3.0: /bin/csh
mdesun% am_menu
mdesun% ☐

Updating DOCUMENTS Directory

To ADD a new document to the Introduction to MDE, you must have two
files. One file contains the formatted document, and the other contains
a table of contents for the document. (See the Programmer's Guide for
a precise description of these files.)

If you wish to continue, type GO.
> GO

Enter a name for the document (up to 20 characters).
> Model_Analyzer

Model_Analyzer will be entered into the Table of Documents.

Enter the full pathname of the file containing the formatted document.
DOCUMENT FILE > /usr/frankel/toprint

Enter the full pathname of the file containing the table of contents for
the document.
TABLE OF CONTENTS FILE > /usr/frankel/toprint

Preparing to copy document and update table of contents.....
Type QUIT if you wish to cancel now, or type GO to continue.
> ☐

Shell Tool 3.0 (CONSOLE): /bin/csh
mdesun% dumpscreen upddocs.screen
☐
```



the feature chosen.

The "Reference" button on the Programmer Assistance screen can be selected to view a Quick Reference guide (Fig. 3.3.6.3). The Quick Reference guide will be displayed in a new window suitable for viewing and scrolling text. This feature is provided as a quick lookup for syntax and usage of the high level procedure calls of the help package. A programmer who is developing an application for the SMDE can use the Reference as a memory refresher and avoid resorting to written documentation.

When not in use, the Programmer Assistance window may be collapsed to its icon representation to conserve screen space.

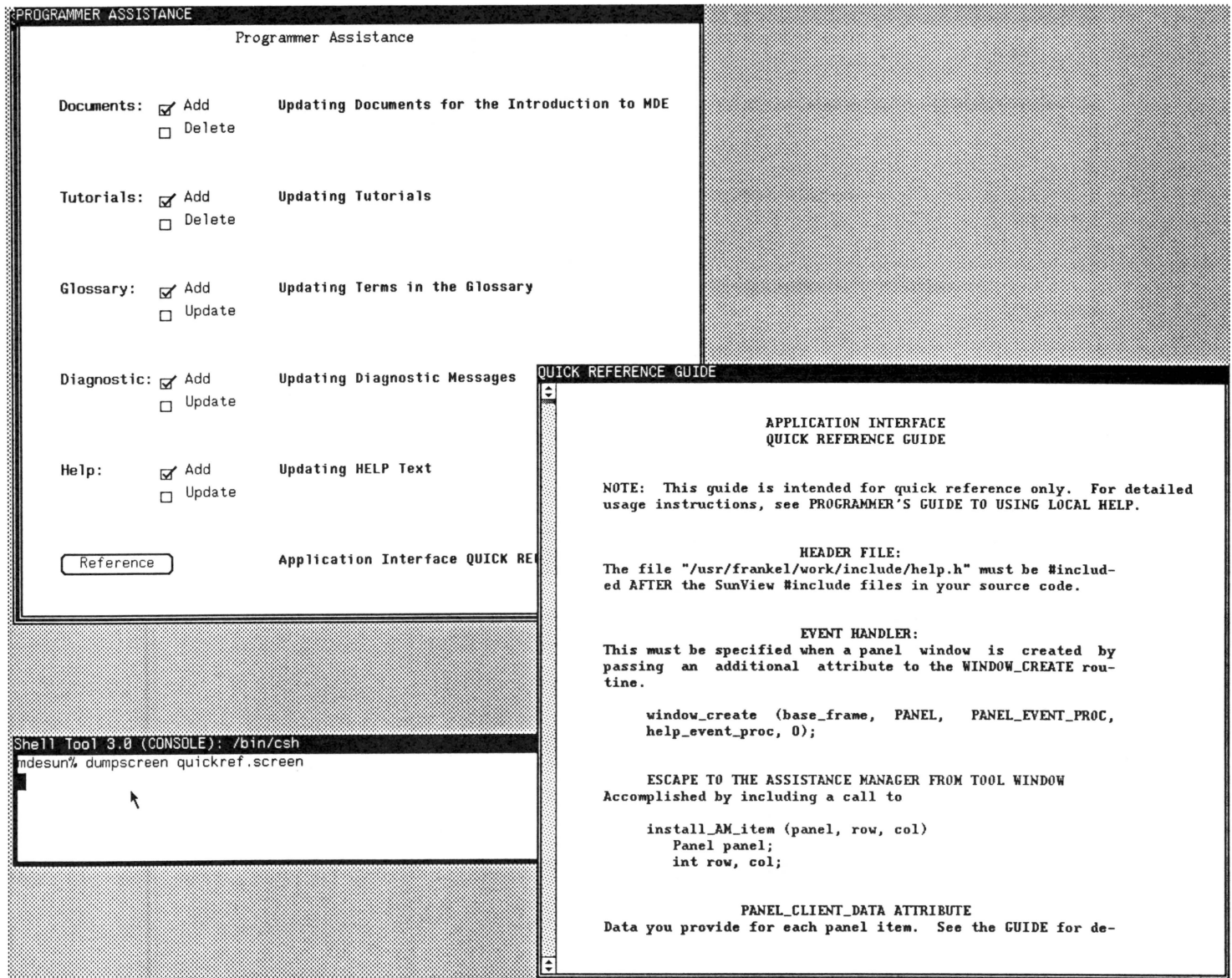
3.4 Implementation Considerations

The Assistance Manager consists of approximately 4500 lines of documented code. The vast majority of this code is written in the C Programming Language [Kernighan and Ritchie 1984], but some parts dealing with file and directory manipulation were written using the Unix shell, sed, and awk [Kernighan and Pike 1984].

Window features were programmed using the SunView application package, which consists of high level routines to create visually communicative interfaces and manage the window system [Sun Microsystems 1986a].

Databases were created using the INGRES database

Fig. 3.3.6.3 Quick Reference Guide Window (top right)



management system [Sun Microsystems 1986b]. EQUeL (Embedded QUeRY Language)/C, made up of extensions to the C Programming Language, was used for retrieval of information for display on the screens [Sun Microsystems 1986c]. The tools used for storing comments and adding data to the help database were developed using the INGRES/FORMS Application Package [Sun Microsystems 1986d].

INGRES is a relational database management system implemented on top of the Unix operating system. As a relational system, INGRES incorporates a high degree of data independence and the possibility of providing a high level and entirely procedure free facility for data definition, retrieval, update, access control, support of views, and integrity verification [Stonebraker et al. 1976].

The EQUeL/C preprocessor enables an application program to run as the front end process to the database. It has the effect of embedding INGRES in a general purpose programming language such as C.

4.0 EVALUATION OF THE PROTOTYPE

A prototype of a software system is a functionally incomplete model of a proposed system, built to demonstrate feasibility or explore potential requirements of a design. The objective of the research described in this thesis is to design a prototype Assistance Manager for the SMDE which provides effective, efficient transfer of information to a user. The prototype represents one possible approach to solving the problem. Evaluating the prototype includes the consideration of how well alternative designs would have addressed the problem under study.

Church et al. [1986] state that two questions are of interest when a software prototype is evaluated:

- * Is the design concept feasible?
- * Is the prototype software an adequate basis for further development?

In this chapter, the prototype of the SMDE Assistance Manager is assessed and evaluated. The evaluation is a design level evaluation only, and is not intended to be exhaustive.

4.1 Assessment Criteria

The criteria used for assessing the prototype Assistance Manager are derived from the design objectives of Section 3.1.2.1. These criteria are presented in Table 4.1.

4.2 Evaluation of the Criteria

The test-bed for studying the design of the Local Help component of the Assistance Manager is a previously proto-

Primarily
Provided For

ITEM	Objective	Modeler	Tool Developer
1	Provide general information for new users.	X	
2	Provide help for an SMDE tool.	X	X
3	Provide definitions of technical terms.	X	
4	Provide tutorial assistance.	X	
5	Provide constantly available, and immediately accessible help.	X	X
6	Provide help that is unobtrusive.	X	X
7	Provide help that is flexible enough for all levels of user.	X	X
8	Provide context-sensitive help.	X	X
9	Provide appropriate access mechanisms.	X	X
10	Provide systematic methods for tool programmers to build help into SMDE tools.		X
11	Provide help that is available in a consistent manner from one tool to another.	X	
12	Serve as an interface between user and the Assistance Manager database.	X	X
13	Provide easy methods for update and expansion of the Assistance Manager database.		X

Table 4.1 Assessment Criteria

typed version of one of the SMDE tools: the Model Generator. The code for the first several screens of the Model Generator was revised to contain calls to the Assistance Manager interface routines. The other components were prototyped as separate processes and integrated at the top level later. By experimenting with the components individually, and also, after they were integrated into a single package, we are able to analyze how well the design meets the assessment criteria. These results are reported below.

Objective 1: Provide general information for new users

-- The Introduction to SMDE provides general information for new users or anyone else who is curious about the SMDE. This information is accessible in such a way that one may invoke it from the Unix command level; therefore, one need not enter the environment to learn more about it.

Objective 2: Provide help for an SMDE tool -- One of the most important objectives of the Assistance Manager is satisfied by incorporating help access mechanisms directly within the tool. These mechanisms make up the Local Help component of the Assistance Manager.

Objective 3: Provide definitions of technical terms -- Definitions of technical terms may be found in the Glossary along with example usage and related information.

Objective 4: Provide tutorial assistance -- The Tutorial component of the Assistance Manager provides an

interface to tutorials for potentially any tool or topic of interest.

Objective 5: Provide constantly available, immediately accessible help -- Help is always "active" in the sense that the user need only select a button to access it. The panel buttons are an integral part of the SMDE user interface, and consequently appear on every tool screen. Their functionality has been expanded to let them be used to request help at any time.

Objective 6: Provide help that is unobtrusive -- The interface design has been done so that the user doesn't "see" the Assistance Manager until he "looks" for it. Help windows pop up and then disappear, and any Assistance Manager screen may be closed or moved to an unobtrusive location on the screen. The context of the user's task is preserved.

Objective 7: Provide help that is flexible enough for all levels of users -- Diagnostic messages and prompts can be requested in progressively greater detail at the discretion of the user. (It should be noted that although the Assistance Manager makes it straightforward for the tool designer and technical writer to give this flexibility, it is ultimately their responsibility to make use of this capability.) In addition, the Assistance Manager provides the Tutorial and Introduction to SMDE for general use, and provides more specific levels of detail in Local Help and

the Glossary. The focus has been to accommodate the modeler who uses the environment frequently, providing a greater level of detail and general information only on request.

Objective 8: Provide context-sensitive help:

Context-sensitive help has been achieved wherever applicable. (Note that the focus of the Introduction to SMDE and the Tutorial, as well as the Glossary in most uses, preclude context-sensitivity as defined in the objectives.) The calls to the Assistance Manager through Local Help are "hard-wired" into the functions of the panel items themselves. It is always possible to determine the user's state and task in this implementation, allowing for a high degree of context sensitivity.

Objective 9: Provide appropriate access mechanisms —

Access to the Assistance Manager is designed around its intended use. When it is serving as tutor or for general information, it is accessible from the Unix command level or from a button on a tool screen. All components except Local Help may be accessed directly or sequentially by top-level menu. Local Help, as expected, is accessible only through a specific tool.

Objective 10: Provide systematic methods for tool programmers to build help into SMDE tools -- This objective is obtained by giving programmers access to a library of routines and definitions which are included as header files and function calls from within application code. (The

functions and header files may only be accessed through the C Programming Language.)

Objective 11: Provide help that is available in a consistent manner from one tool to another -- The same library that meets Objective 10 ensures that access to the Assistance Manager database will be consistent across all applications. The centralized control maintained by the library guarantees uniformity in the user interface.

Objective 12: Serve as interface between user and the Assistance Manager database -- The Assistance Manager database is only accessible through Assistance Manager mechanisms. Updates and additions to the database can only be made after authorization has been granted by the system administrator. This is enforced by restricting database modifications to users who supply the appropriate access code once the Programmer Assistance tool has been invoked.

Objective 13: Provide easy methods for update and expansion of the Assistance Manager database -- All database updates may be made through the Programmer Assistance component of the Assistance Manager. Through a series of prompts and menus, authorization is verified, and the update proceeds through a controlled series of data entry "forms" -- screens that are customized for a particular data entry operation. Data already present in the database may be edited or deleted, and new data may be added by the same conventions.

4.3 Design Alternatives

Among the possible alternatives to the design of the Assistance Manager, two in particular appear most reasonable and are considered here. A commercially available add-on system could have been adapted for use with the SMDE. Such a system can incorporate excellent help texts and completeness of information; however, major disadvantages exist. A lack of context-sensitivity is painfully evident. Because an add-on system has no mechanism for tracking the user's current state of interaction, it cannot use such information to provide timely and appropriate assistance. Another disadvantage is the lack of unobtrusiveness. A user has no choice but to interrupt this present task and go off to request help through a separate mechanism. Although it is possible to step through a hierarchy of progressively more specific information, the information may be accessed in one way only; namely, through a top-down approach to the topic under study.

An add-on system could have met the requirements for expandability and maintainability, but fails in the area of flexibility since the same help text is presented to all users.

A second alternative to the design of the Assistance Manager was a completely menu-based system. A menu-based system potentially offers more of the desirable

characteristics than an add-on system. A degree of context sensitivity could be provided by using a network such as the one described in [Robertson et al. 1981]. Consistency is obtained by providing access to help in the same manner across the entire system. The texts of the menu-based system can be prepared so that they are complete and understandable, and asking for help can be as easy as pointing to an item on the menu and clicking a button.

One disadvantage to the menu-based system is the lack of flexibility. As in a commercially available help package, any user, no matter how experienced, must page through a hierarchical organization of information until the specific information is located. The system forces the user to think structurally rather than contextually. The result is that a quest for assistance is often put off simply because the user is faced with the distracting and time-consuming task of stepping through the menu interface.

Although menus are used extensively throughout the Assistance Manager prototype, no menu is ever more than two layers deep. The menus are often "pop-up", and can be bypassed altogether. Most importantly, menus are restricted to the components of the Assistance Manager where a user is more likely to "browse" for information, such as the Glossary and the top-level Assistance Manager menu. For a user who needs immediate and specific help within an SMDE tool, the Local Help features provide this access without

ever forcing the user to make menu selections.

4.4 Design Limitations

As was expected, the prototyping process exposed some minor flaws in the design, as well. The Glossary component is somewhat cumbersome, and could benefit from some streamlining in its interface. The "lookup" feature is of questionable value, since the same functionality is provided through the "browse" feature.

Text displayed in the Tutorial and Introduction to SMDE viewing windows is somewhat difficult to read. Experimentation with different font styles and sizes would be beneficial. Maximum benefit could be gained from these components if they included more text location and retrieval features, similar to the capabilities of the DOMAIN/DELPHI system [Orwick et al. 1986]. The Tutorial could feasibly be somewhat context-sensitive if, when invoked from an SMDE tool, it began execution in the tutorial for the calling tool.

A further limitation exists in providing button help for some of the panel item types on a SunView window. A few of these, such as slider and toggle types, are not selectable by the mouse and have no mechanism to incorporate help information.

Hands-on interaction with the prototype will doubtlessly give further insight into the usefulness of the

Assistance Manager features. Programmer experimentation with the Assistance Manager library of routines is necessary to further evaluate its ease of use and flexibility. Recommendations to this effect are discussed in Chapter 5.

In summary, however, the Assistance Manager prototype demonstrates the desirable characteristics of online assistance without the disadvantages inherent in more traditional forms of online help. The design concept is feasible and provides an adequate basis for further development and integration with future SMDE tools.

4.5 The Choice of INGRES as a Database Management System

The choice of a database management system (DBMS) affects all SMDE tools which require the services of a DBMS. The prototype of the Assistance Manager is useful as a vehicle for testing our choice of INGRES, and of the relational model in general, for the SMDE.

The two most important requirements for the DBMS used by the Assistance Manager are portability and support for variable length text fields. The portability requirement is universal throughout the SMDE. The support for variable length text fields is a requirement most evident within the Assistance Manager, which relies almost exclusively on storage and manipulation of text strings.

We find our choice of INGRES to be satisfactory for the prototype of the Assistance Manager, although we

encountered some significant problems with the INGRES software. The problems we faced were those typical of a large software system ported to a new machine architecture and operating system. Although we suffered several setbacks, all implementation problems were eventually solved with vendor support.

Performance problems exist with INGRES which may make it undesirable for use in a final system; however, this disadvantage must be weighed against the significant advantages supplied by a good screen and forms manipulation capability.

The relational model serves as an easy to understand data model which enables the data to be conceptualized as tables of text fields uniquely identified by integer keys. It should be noted, however, that other data models (e.g., the hierarchical model) may have proved equally successful in the implementation of the Assistance Manager.

5.0 SUMMARY AND CONCLUSIONS

5.1 Summary

This thesis describes the features and capabilities of the Assistance Manager for the Simulation Model Development Environment and the research which justifies the design methods and choices. This research has emphasized a single goal: to prototype an online assistance system for the SMDE which provides effective and efficient transfer of information to a user of the system. Aspects of existing help systems are examined, and research trends are explored to see if they merit consideration in the system's design. Decisions on system functionality and features are made on the basis of how well they meet the characteristics considered necessary for successful online assistance.

A classification of help systems is identified. Desirable characteristics for online assistance systems are identified and used as guiding principles for the design of the Assistance Manager. Existing help systems are shown not to provide all forms of assistance consistently to all levels of user.

An effective method for composing and organizing assistance and diagnostic messages is designed which automates the construction and maintenance of help scripts. Thus, the composition of help text is separated from the implementation of the help system. The Assistance Manager is prototyped, and the prototype is evaluated against

design alternatives. The Assistance Manager is found to uphold and fulfill the design requirements to an extent which justifies its continued support within the Simulation Model Development Environment. The Assistance Manager meets the design requirements by consistently providing help that is constantly available, context sensitive, understandable, unobtrusive, complete, and flexible, thereby (1) improving a user's understanding of a system, (2) minimizing the likelihood of user errors, (3) reducing the amount of information a user must remember to perform a task, and (4) enabling users with widely different levels of experience to use the same software system.

5.2 Recommendations for Future Research

Several areas related to the Assistance Manager deserve further consideration. The first area of concentration should be in constructing experiments to discover how well the Assistance Manager responds to practical usage. Such experiments might consist of giving users a progression of tasks to perform within the SMDE, from simple to more complex, and monitoring to what extent the Assistance Manager is used. These experiments could help determine whether the Assistance Manager effectively "encourages" users to seek help from the assistance system.

A primary reason for prototyping is to create a basis for further refinements. The prototype gives the developer

a means to experiment with new ideas while the system is still evolving. The Assistance Manager must be judged with a critical eye directed towards its user interface and functionality. Suggestions must be made and, where necessary, compromises reached. A case in point is the status subwindow of a tool which is used to display prompts and diagnostics. The window currently is placed on the lower portion of the viewing area. This seems the most natural placement for the status area during the design phase. However, in practice, the status area falls just out of the immediate visual range, resulting in missed diagnostics and unheeded prompts. A better solution would be to place the status window at the top of the tool, or to flash the display momentarily.

Similarly, design alternatives may appear preferable during the prototyping phase, but glaringly deficient when put into practical use. Often these deficiencies will be noticed only after the system has been used for some time. Another aspect of the Assistance Manager which can benefit from practical application is the programmer interface. More data needs to be collected to determine what additional components may be required to ease the burden on the software system developer who must use the Assistance Manager package. These questions can best be answered after the interface routines have been exercised by applications

programmers.

A related issue concerns guidelines and automated support for help text composition and hard copy generation. The Assistance Manager at present contains no facility to integrate the database scripts to generate an organized user manual. The report generation capabilities of INGRES are useful for little more than producing a printed version of the database contents.

Tools which could further ease the burden on script writers include a spelling checker and a tool to automate text formatting.

Perhaps one of the most obvious areas of enhancement is the provision of some sort of system "intelligence", perhaps in the form of an advice-giving expert system. Efforts to provide such a feature could possibly detract from the generality we have tried to maintain in the Assistance Manager, in that expert system features are typically domain specific. An alternative would be to add an expert-system shell which could be tuned by the target application developers.

5.3 Conclusions

The research clearly indicates that most online assistance systems fall short of their potential despite the current emphasis on systems that are human-engineered. There has been little attempt on the part of implementors to

build on the past successes of others; designers have consequently repeated many mistakes and been doomed to the same failures.

Today's software and hardware technology can inspire imaginative ways of integrating a sophisticated online assistance package with a software system. Online assistance should never have to be implemented in an ad-hoc or post-hoc fashion.

Perhaps the most important question remaining for software engineers is how to persuade the user to make use of online assistance. The truest test of a help system is whether it is used, and whether its use is an effective aid to productivity.

BIBLIOGRAPHY

- Balci, O. (1986), "Requirements for Model Development Environments," Computers and Operations Research 13, 1, (Jan. - Feb.), 53-67.
- Bergman, H. and J. Keene-Moore (1985), "The Birth of a Help System," In Proceedings of the 1985 ACM Annual Conference, (Denver, Co.), ACM, New York, N. Y., pp. 289-295.
- Blake, T. (1986), "The Art and Science of User Interface Design," Tutorial Presented at the Conference on CHI '86 Human Factors in Computing Systems (Boston, Mass., Apr. 14-17).
- Borenstein, N. S. (1985), The Design and Evaluation of Online Help, Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, Pa., Apr.
- Bramwell, B. (1983), "BROWSE: An On-Line Manual and System Without an Acronym," ACM SIGDOC Asterisk 9, 4 (Apr.), 7-11.
- Butler, T. and L. Kennedy (1985), "The AT&T Unix System Help Facility," Unix/World 2, 6 (June), 81-83, 102.
- Card, S. K., W. K. English, and B. J. Burr (1978), "Evaluation of Mouse, Rate-Controlled Isometric Joystick, Step Keys, and Text Keys for Text Selection on a CRT," Ergonomics 21, 8 (Aug.), 602-613.
- Carlson, P. A. (1983), "User-Programmer Dialogue: Guidelines for Designing Menus and Help Files for Interactive Computer Systems," NASA Technical Memorandum 84980, Goddard Space Flight Center, Greenbelt, MD.
- Carroll, J. M. and C. Carrithers (1984), "Training Wheels in a User Interface," Communications of the ACM 27, 8 (Aug.), 800-806.
- Chin, D. N. (1986), "User Modeling in UC, the Unix Consultant" In Proceedings of CHI '86 Human Factors in Computing Systems, (Boston, Mass., Apr. 14-17). ACM, New York, N.Y., pp. 24-28.
- Church, V. E., D. N. Card, W. W. Agresti, and Q. L. Jordan (1986), "An Approach for Assessing Software Prototypes", ACM Sigsoft Software Engineering Notes 11, 3 (Apr.), 1-12.

- Clark, I. A. (1981), "Software Simulation as a Tool for Usable Product Design," IBM Systems Journal 20, 3 (Apr.), 272-293.
- Connolly, T., A. Bradford, R. Grice, and J. Steipp (1985), "Issues Concerning the Development and Use of Online Information," ACM SIGDOC Asterisk 11, 1 (Jan.), 21-30.
- Digital Equipment (1980), VAX/VMS Command Language User's Guide, Digital Equipment Corporation.
- Estrin, G., R. S. Fenchel, R. R. Razouk, and M. K. Vernon (1986), "SARA (System Architects Apprentice): Modeling, Analysis, and Simulation Support for Design of Concurrent Systems," IEEE Transactions on Software Engineering SE-12, 8 (Aug.), 293-311.
- Fenchel, R. S. And G. Estrin (1982), "Self-Describing Systems Using Integral Help," IEEE Transactions on Systems, Man and Cybernetics 12, 2 (Feb.), 162-167.
- Gait, J. (1985), "An Aspect of Aesthetics in Human-Computer Communications: Pretty Windows," IEEE Transactions on Software Engineering SE-11, 8 (Aug.), 714-717.
- Hayes, P., E. Ball, and R. Reddy (1981), "Breaking the Man-Machine Communication Barrier," Tutorial: Software Development Environments, IEEE Computer Society Press, Silver Spring, Md., 302-312.
- Hodgson, G. M. and S. R. Ruth (1985), "The Use of Menus in the Design of On-Line Systems: A Retrospective View," SIGCHI Bulletin 17, 1 (Jan.), 16-21.
- Houghton Jr., R. C. (1984), "Online Help Systems: A Conspectus," Communications of the ACM 27, 2 (Feb.), 126-133.
- IBM (1983), VM/System Product CMS Primer, IBM Corporation.
- Kernighan, B. W. and R. Pike (1984), The Unix Programming Environment, Prentice-Hall, Englewood Cliffs, N. J.
- Morland, D. V. (1983), "Human Factors Guidelines for Terminal Interface Design," Communications of the ACM 26, 7 (July), 302-312.
- Mozeico, H. (1982), "A Human/Computer Interface to Accommodate User Learning Stages," Communications of the ACM 25, 2 (Feb.), 100-104.

- Nakatani, L. H. (1983), "Soft Machines: A Philosophy of User-Computer Interface Design," In Proceedings of the CHI '83 Human Factors in Computing Systems, (Boston, Mass., Dec. 12-15), ACM, New York, N.Y., pp. 19-23.
- Nakatani, L. H., D. E. Egan, L. W. Ruedisueli, et.al. (1986), "TNT: A Talking Tutor 'N Trainer for Teaching the Use of Interactive Computer Systems," In Proceedings of the CHI '86 Human Factors in Computing Systems, (Boston, Mass., Apr. 14-17), ACM, New York, N. Y. pp. 29-34.
- Norman, D. A. (1983), "Design Rules Based on Analyses of Human Error," Communications of the ACM 26, 4 (Apr.), 254-258.
- O'Malley, C., P. Smolensky, L. Bannon, E. Conway, J. Graham, J. Sokolov, and M. L. Monty (1983), "A Proposal for User-Centered System Documentation," In Proceedings of the CHI '83 Human Factors in Computing Systems, (Boston, Mass., Dec. 12-15) ACM, New York, N.Y. pp. 282-285.
- Orwick, P., J. T. Jaynes, T. R. Barstow, L. S. Bohn (1986), "DOMAIN/DELPHI: Retrieving Documents Online," In Proceedings of the CHI '86 Human Factors in Computing Systems, (Boston, Mass., Apr. 14-17), ACM, New York, N.Y., pp.114-121.
- Price, L. A. (1981), "Using Offline Documentation Online," SIGSOC Bulletin (ACM) 13, 15-20.
- Price, L. A. (1982), "Thumb: An Interactive Tool for Accessing and Maintaining Text," IEEE Transactions on Systems, Man and Cybernetics 12, 2 (Feb.), pp. 155-161.
- Relles, N. (1979), The Design and Implementation of User-Oriented Systems, Ph.D. Thesis, University of Wisconsin-Madison, Aug.
- Relles, N. and L. A. Price (1981), "A User Interface for On-Line Assistance," In Proceedings of the 5th International Conference on Software Engineering, (San Diego, Ca., Mar.) IEEE, Piscataway, N.J. pp. 400-408.
- Relles, N., N. K. Sondheimer, and G. P. Ingargiola (1981), "A Unified Approach to Online Assistance," Proceedings of the 1981 National Computer Conference, (Arlington, Va.), AFIPS Press, pp. 383-388.

- Robertson, G., D. McCracken and A. Newell (1981), "The ZOG Approach to Man-Machine Communication," Intl. Journal of Man-Machine Studies 14, 461-488.
- Roberts, R. (1970), "HELP: A Question Answering System," In Proceedings of the 1970 Fall Joint Computer Conference, (Arlington, Va.) AFIPS Press, pp. 547-554.
- Roemer, J. M. and A. Champanis (1982), "Learning Performance and Attitudes as a Function of the Reading Grade Level of a Computer-Presented Tutorial," In Proceedings of the Conference on Human Factors in Computing Systems, (Gaithersburg, MD, Mar. 15-17), ACM, Washington D.C. Chapter, pp. 239-244.
- Rothenberg, J. (1979), "Online Tutorials and Documentation for the SIGMA Message Service," In Proceedings of AFIPS National Computer Conference 48, pp. 863-867.
- Schneider, M. L. (1982), "Models for the Design of Static Software User Assistance," In: A. Badre and B. Shneiderman (editors), Directions in Human-Computer Interaction, Ablex Publ. Co., pp. 137-148.
- Shneiderman, B. (1984), "Response Time and Display Rate in Human Performance with Computers," ACM Computing Surveys 16, 3 (July), 265-285.
- Smith, D. C., C. Irby, R. Kimball, and B. Verplank (1982), "Designing the STAR User Interface," Byte, (Apr.), 242-282
- Sondheimer, N. K. and N. Relles (1982), "Human Factors and User Assistance in Interactive Computing Systems: An Introduction," IEEE Transactions on Systems, Man, and Cybernetics SMC-12, 2 (Feb.), 102-106.
- Standards and Guidelines Information Exchange Group, Toronto (1985), "Online Documentation Development Guidelines," ACM SIGDOC Asterisk 11, 1 (Jan.), 7-19.
- Stoddard, M., K. Berkbigler, B. Wheat, and E. Peter (1985), "User Behavior Upon Introduction of a Network Help System," SIGCHI Bulletin 16, 3 (Mar.), 25-31.
- Stonebraker, M., E. Wong, and P. Kreps (1976), "The Design and Implementation of INGRES," ACM Transactions on Database Systems 1, 3 (Sept.), 189-222.
- Sun Microsystems (1986), SunView Programmer's Guide, Sun Microsystems, Inc., Mountain View, CA, Feb.

- Sun Microsystems (1986), SunView System Programmer's Guide, Sun Microsystems, Inc., Mountain View, CA, Feb.
- Sun Microsystems (1986), SunINGRES/Equel/C, Sun Microsystems, Inc., Mountain View, CA, May.
- Sun Microsystems (1986), SunINGRES/ABF (Application-By-Forms) User's Guide, , Sun Microsystems, May.
- Teitelman, W. (1984), "A Display Oriented Programmer's Assistant," In: Barstow, Shrobe, and Sandewall (editors), Interactive Programming Environments, McGraw-Hill Publishing Co., pp. 241-287.
- Walker, J. (1986), "Online Documentation and HELP Systems," Tutorial Presented at the Conference on CHI '86 Human Factors in Computing Systems, (Boston, Mass., Apr. 14-17).
- Yestingsmeier, J. (1984), "Human Factors Considerations in Development of Interactive Software," SIGCHI Bulletin 16, 1 (Jan.), 24-27.

Attention Patron:

Page 79 repeated in numbering

APPENDIX PROGRAMMER'S GUIDE TO USING LOCAL HELP

1. Adding HELP to the User Interface

Any application programmer developing tools under SunView Version 3.0 may automatically include HELP with his program. All that is necessary is to "#include" the header files described below, and to provide some additional information for the SunView interface. Read on for details.

1.1. Providing Escape to the Assistance Manager

Somewhere in your main level routine, you must make provision for adding an item to allow a user to escape to the Assistance Manager tools. This is enabled by including a call to the HELP package. The correct syntax is:

```
install_AM_item (panel, row, col)
    Panel panel;
    int row, col;
```

The item will subsequently appear on your panel window at the locations specified by col and row (default col = 65, row = 1). If you are using more than one panel, and you want to include the item on other panels, then *install_AM_item* must be called for each panel.

1.2. Header Files

There is one header file to include with your source code: (usually, the "#include" statement for this file will be placed in your source file after the #include files necessary for SunView)

```
#include "usr/frankel/work/include/help.h"
```

This header file contains the definitions and routines necessary to incorporate HELP for panel items.

1.3. Additional Information for the SunView Interface

The header file defines the data structures and procedures to be used for HELP, but you must explicitly inform the SunView Interface of the changes that will occur. Fortunately, this is a simple matter involving few lines of extra code.

1.3.1. Changes in the Event Handling Mechanism

Because using HELP involves changing the way input from the mouse is handled, the default *panel_event_proc* must be replaced. This is done at the time you create the panel window, by passing an additional attribute to the *window_create* routine as shown below:

```
window_create (base_frame, PANEL,
    PANEL_EVENT_PROC, help_event_proc,
    ... /* any other window attributes go here */
    0);
```

1.3.2. Storing Information with PANEL_CLIENT_DATA

When HELP wishes to work with one of your application's panel items, it needs information on the type of item, the help message you want to display, etc. The only way HELP can get to this information is for you to provide it explicitly. This is done with the *PANEL_CLIENT_DATA* attribute of the *panel_create_item* routine.

HELP expects you to store a pointer to a struct of type *Item_data* with each panel item you create. The fields of this structure are filled in with information that HELP will need. An

example will help clarify this usage.

```
int init_panel() /* Example routine creating items on a panel. */
{
    int i;
    Item_data *my_data[2]; /* If we create two panel items, we need a */
                          /* struct for each. */

    /* First, we must allocate storage for each structure. */

    for (i = 0; i < 2; i++)
        my_data[i] = (Item_data *) malloc (sizeof (Item_data));

    /* Now, as each panel item is defined, set the data to be stored
     * with each item. First comes an item of type PANEL_BUTTON:
     */

    my_data->help_item_type = BUTTON; /* This can be BUTTON, MESSAGE,
                                     * TOGGLE, CHOICE, or SLIDER.
                                     */

    my_data->item_label.label = /* if label is an image, then a pointer
                              * to the image is assigned here --
                              */

    my_data->item_label.string = /* else if label is a string, then the
                              * string is assigned here.
                              */

    my_data->utype = /* either IMG_LABEL if label is image, or
                   * STR_LABEL if label is a string */

    my_data->text_key = /* Integer value of help key as obtained from
                      * database (see below for descriptions.
                      */
}
```

2. Updating the HELP Database to Contain Help Messages for Your Program

The instructions above add the necessary changes to the video display and user interface. The task of composing help text and adding this text to the database must be done separately. This section contains instructions to accomplish this task.

As you work with SunView to create the interface to your application, keep in mind that you will be providing HELP for each panel item you display on the screen. You have already been storing text_keys with each item via the panel_client_data attribute. To develop and debug your program, you can use dummy #DEFINE statements to #DEFINE integer values for these keys. When the time comes to integrate the HELP text with your interface, you will replace the integer constants you've been using with values that will actually index the HELP database.

When you are ready to add HELP text to the database, you can do so by running the Assistance Manager inside the SunTools environment. You are presented with a menu of choices, from which you should select "Programmer Assistance". The "Programmer Assistance" component of the Assistance Manager allows you to add, delete, and update all the HELP databases for which

you have permission.

For adding the HELP text for your application, you will want to select the item labeled "Updating Help Text". The next screen that appears puts you in direct access to the database. HELP text is typed in the fields displayed. You should run the Assistance Manager now to become familiar with these techniques.

Note that the field titled "MSGNUM" always appears with a value. The database automatically updates and assigns these keys, guaranteeing that the number will be unique to your particular entry. These values will replace the dummy values you previously #DEFINED for your text_keys.

3. Error Diagnostics

If you are expecting some form of input from the tool user, then it is quite likely your application must take into account error conditions. For example, the user may request access to a non-existent file, or a protected database. For these occasions, it is strongly recommended that your tool window contain a "message" area to display error messages and user prompts. If you wish to include a separate panel for your message area, you can make use of the diagnostic package provided through the Assistance Manager to display diagnostics. A call to *install_message_area* after you've created the panel subwindow will initialize the display area for you.

To display a particular message in the message area, use the procedure

```
errmsg (win_name, string1, string2, errnum)
Panel win_name;
char *string1, string2;
int errnum;
```

To display simple prompts, you can call *errmsg* with the name of the window message area, and up to two strings (each with a maximum of 70 chars). If the *errnum* parameter is 0, the routine knows this is a simple prompt and will not attempt to access the database for diagnostic information.

If, on the other hand, you wish to display diagnostics in response to a user error, *string1* and *string2* should be null, and *errnum* should be a key to a message in the diagnostic database. (These integer keys are determined similar to the method used with HELP text_keys. See the next section for details.)

When the diagnostic appears in the window, an "Explain" button also appears. Thus, you may provide many levels of diagnostics in response to a user's selection of the "Explain" button, from simple and concise to detailed and complex. Because these messages are linked together in the database, you only need to call *errmsg* for the first message. The diagnostic package takes care of the rest.

A routine

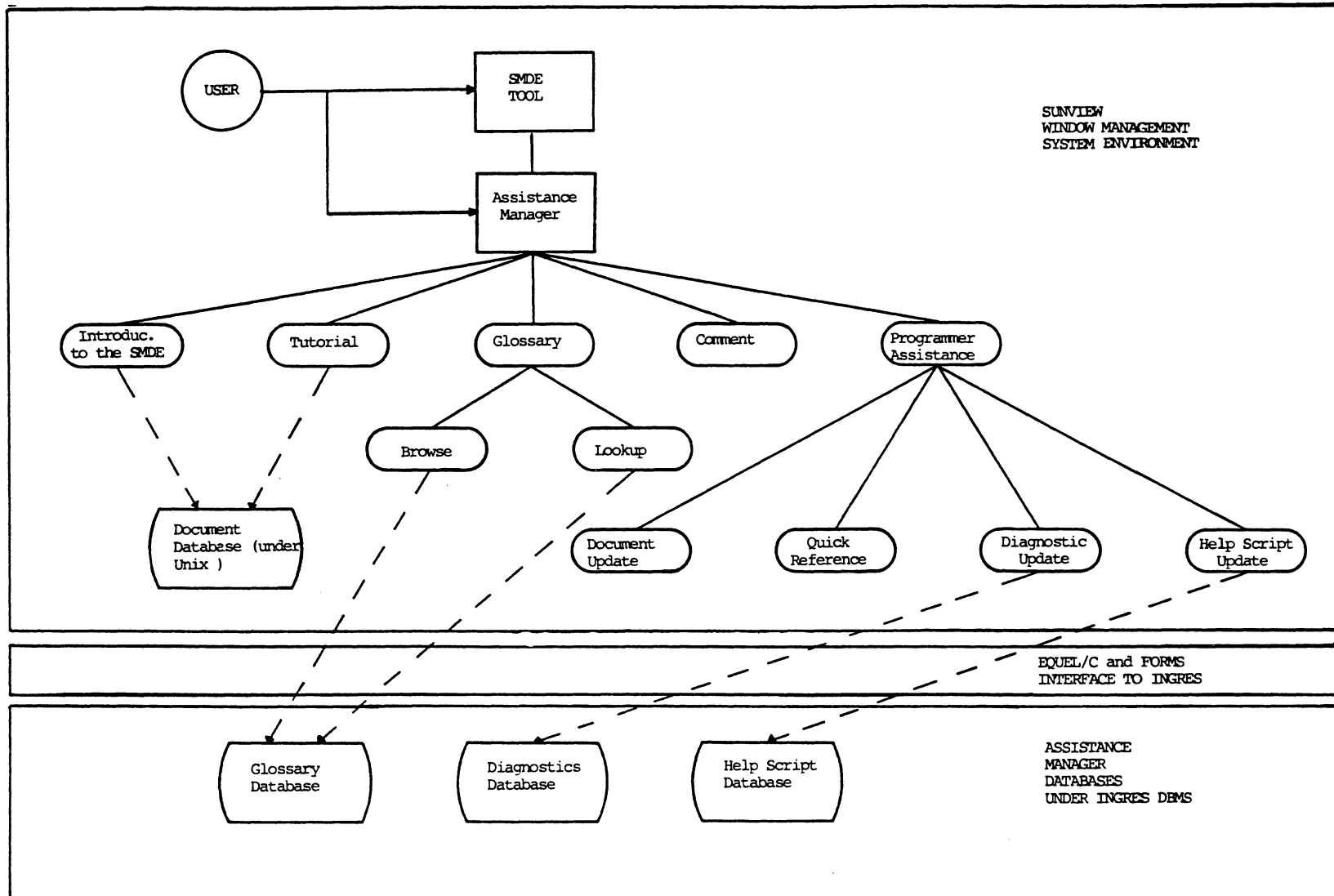
```
remove_message (win_name)
Panel win_name;
```

is provided to erase the message display.

3.1. Updating the Diagnostics Database to Contain Diagnostics for Your Program

Just as when you updated the HELP database, you must update the Diagnostic Database via the Assistance Manager. This time, you should select "Diagnostics" from the Programmer Assistance second-level menu. You will have a direct access to the database, and the next key value available for use will appear in the "errnum" field. (This will be the value you pass to the routine *errmsg*.)

APPENDIX 2: Architecture of the SMDE Assistance Manager



**The vita has been removed from
the scanned document**