Universal Software Documentation via Dynamic Help

ASQB-GM-91-02
May 1991

AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800
The purpose of this project was to develop a documentation strategy that serves both the needs of initial documentation and those of Dynamic Help. Implementing Dynamic Help in the final software product will benefit the software user; the act of incorporating Dynamic Help concepts into the requirements document may actually assist the requirements writer by insuring that certain vital questions are asked and answered, but it will require that requirements be expressed partly through formal data structures.
This research was performed for the Army Institute for Research in Management Information, Communications, and Computer Sciences (AIRMICS), the RDTE organization of the U.S. Army Information Systems Engineering Command (USAISEC). This research report is not to be construed as an official Army position, unless so designated by other authorized documents. Material included herein is approved for public release, distribution unlimited. Not protected by copyright laws.

THIS REPORT HAS BEEN REVIEWED AND IS APPROVED

s/ James Gant
Division Chief
MISD

s/ John R. Mitchell
Director
AIRMICS
Universal Software Documentation via Dynamic Help

Individual Research Report

by

Walter S. Barge, II
Captain, United States Army

Project Course ISyE 8601A3
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332-0205

Course Advisor:
Donovan Young

May 31, 1991
# Table of Contents

1. Requirements Documentation and Dynamic Help.................................1  
   1.1 Why Dynamic Help?.............................................................................2  
   1.2 Definition and Discussion.................................................................2  
2. Current Research on Dynamic Help.....................................................6  
3. Common Practices in Requirements Documentation............................7  
4. Matching Dynamic Help to Requirements Documentation....................11  
   4.1 Context...............................................................................................14  
   4.2 Navigation.........................................................................................14  
   4.3 Meaning............................................................................................15  
   4.4 Domain..............................................................................................15  
   4.5 Content.............................................................................................15  
   4.6 Format...............................................................................................16  
   4.7 General.............................................................................................16  
5. Demonstration of Universal Documentation........................................17  
   5.1 Illustrative Example  
      University Registration System..........................................................17  
   5.2 Registration Function.......................................................................22  

# List of Figures

Figure 1. The Eight Generic Dynamic Help Sentence Types.................3  
Figure 2. Interface Clarity for Novice/Intermittent Users......................5  
Figure 3. Example Screen for the University Registration  
      System................................................................................................22  
Figure 4. Example Schedule Window....................................................23
1. Requirements Documentation and Dynamic Help

The creation of a comprehensive requirements document is the true foundation for the remainder of the software development process. A poor requirements document will often "snowball" into a poor product; conversely, good requirements documentation improves the project's chances for success.¹

Dynamic Help is a method of building on-line user help messages at the time they are requested by the software user. The Dynamic Help messages are assembled from both stored phrases and the values of available state variables. Because the messages are built only by request they are economically and syntactically more efficient than stored text help messages.²

Good requirements documentation reduces development cost and improves the final software product; Dynamic Help improves the efficiency and usefulness of on-line user help. This paper will show how the integration of Dynamic Help concepts into requirements documentation can result in a software product that is easier both to develop and to use than a product lacking this combination.

Our goal is to develop a documentation strategy that serves both the needs of initial documentation and those of Dynamic Help. Implementing Dynamic Help in the final software product will benefit the software user; the act of incorporating Dynamic Help concepts into the requirements document may actually assist the requirements writer by insuring that certain vital questions are asked and answered.
but it will require that requirements be expressed partly through
formal data structures.

1.1 Why Dynamic Help?

The term Dynamic Help originated at Georgia Institute of
Technology during research supporting the US Army's Installation
Support Module (ISM) project. The three broad goals of the ISM
project are 1) to improve (or create) horizontal and vertical DBMS
integration at the individual installation level, 2) to consolidate
duplicated functions and actions on an installation, and 3) to improve
the resulting system's accessibility to novice and intermittent users.3

Clearly, the third ISM goal can only be achieved "if the software
modules have user interfaces that are reasonably consistent, easy to
learn, and [are] designed for the needs of the novice and intermittent
users."4 Dynamic Help seeks to improve the user interface through an
efficient architecture that provides help messages tailored to the level
of novice and intermittent users. Because Dynamic Help is invoked by
the user, its availability will not impede the experienced user who can
complete the tasks at hand without its use.

1.2 Definition and Discussion

Dynamic Help is a message building strategy that is a combination
of contextual information and accessible dynamic state variables. Help
messages are assembled by the concatenation or interleaving of stored
text phrases and variable system data. The variable data is retrieved
only after the request for help by the user. By operating in this
manner, the help message is fully descriptive and current without the
need for storing the entire text of the message.
Researchers concerned with Dynamic Help (Young, Wolven, Haines) have realized that much of the material retrieved by a Dynamic Help module when building its messages -- names of objects, descriptions of functions, domains of variables, formats of data -- is material that should be generated as part of requirements documentation; however, for it to be automatically usable by the Dynamic Help module would require control of the structures, syntax and format of the data by which the requirements are expressed.\(^5\)

For example, a list of locations, or screens, can constitute the data that the Dynamic Help module would query to answer the question "Where am I?". Similarly, lists of commands and functions can constitute the data needed for the Dynamic Help module to answer the question "What can I do here?".

The heart of Dynamic Help is a set of eight generic sentence-types which contain phrases and logically placed slots for the insertion of system state variables (Figure 1). The generality of the sentences allows help messages to be built at virtually every point during the use of the software.

In order for Dynamic Help to improve the interface clarity for novice and intermittent users it must address the questions shown in Figure 2. The answers to these questions are the basis for construction of the eight generic Dynamic Help sentences.

Dynamic Help is not applicable to all types of software designs. Designs and applications which possess a high degree of data discipline, such as those based on a database management system (DBMS), can provide Dynamic Help with the ability to query contextual
Context Ready to <accomplish a particular task>.

Navigation (global and local)
- Global: To return to <a previous location><perform this action>.
- To move to <the next location><perform this action>.
- Local: (form depends on screen design; concerns management of cursors, objects and commands on the current screen)

Meaning <definition of the current task or field>.

Domain Acceptable entries are <list of descriptors>.

Content <a list of legal entries>.

Format <an example of a correct entry>.

General For general documentation press the F1 key.

Figure 1. The Eight Generic Dynamic Help Sentence Types

information and retrieve state variables. Databases have the additional advantage that context data (names of screens, names of data entry variables, etc.) is often stored in relational tables and this can be accessible in the same way as state variables. Designs with ad hoc or non-standard data structures are not good candidates for Dynamic Help.

Systems with a highly defined functional geography give Dynamic Help a detailed context from which to work. Protocols which specify active objects and active commands provide narrowly defined “places” in the system that can be identified and referred to by Dynamic Help.

Examples of systems like this are DBMS-based systems, interactive scheduling programs and any system where such things as cursor locations, highlighted objects, highlighted commands, or rigid procedures provide highly specific contexts.
<table>
<thead>
<tr>
<th>Where am I? (location name, functions supported)</th>
</tr>
</thead>
<tbody>
<tr>
<td>What [commands] are available?</td>
</tr>
<tr>
<td>What does this [object] [entry] mean?</td>
</tr>
<tr>
<td>How can I get to the [ordinarily next] [last previous] [next higher] location?</td>
</tr>
<tr>
<td>How can I undo [correct] this [command] [object] [entry] ?</td>
</tr>
<tr>
<td>What is the [format] [legal content] of this entry?</td>
</tr>
<tr>
<td>How can I get to the [ordinarily next] [last previous] [next higher] [command] [object] [entry] ?</td>
</tr>
</tbody>
</table>

Figure 2. Interface Clarity for Novice/Intermittent Users

By contrast, a word processing program is not a good candidate for Dynamic Help. It lacks an easily manipulated data structure from which to extract system state variables. The local context is very vague and there is little or no information available on the user's intent; procedures are flexible rather than rigid.

The use of confirm cycles and closure discipline will increase the benefits of Dynamic Help. Providing the user with the ability to choose a command first, then invoke it, gives him the chance to invoke Dynamic Help. Premature closure (the acceptance of an input datum
upon merely filling its field, or invocation of a command upon merely selecting it) takes away the opportunity to request help at a time when it may be needed the most.

2. Current Research on Dynamic Help

Recent research has focused on demonstrating the efficacy of Dynamic Help and on determining its applicability to software now used by the U.S. Army. As part of an ISM pilot-project, Dynamic Help was added to the Army's existing Automated Central Issue Facility System (ACIFS) software. A recent Master's Thesis by Captain Renée S. Wolven at Georgia Tech experimentally tested the CIF software with the added Dynamic Help. Her results show that Dynamic Help significantly improved the user interface by reducing the number of dead ends (points where the user could not proceed) and by reducing the number of errors. Dynamic Help did not save time because the act of accessing and reading the messages itself takes time. Captain Wolven found that when invoking Dynamic Help the users most often needed help with commands or data entry. She recommended improving the sentence order in Dynamic Help messages to reduce reading time by putting the Navigation sentence as the next to last sentence. This change allows the Dynamic Help information explaining user commands and data entry to be read first. This modification has been incorporated into examples later in this paper.

The stored text portions of the Dynamic Help messages used in the CIF software were generated manually by Captain Wolven. Her thesis showed the advantages of using Dynamic Help in the ACIFS
software. In another Master's Thesis at Georgia Tech, Captain Stanley K. Haines demonstrated how the stored text portions of the Dynamic Help messages for CIF could be generated automatically by querying a hypothetical documentation database. The documentation database contained items and phrases that would be a part of any good requirements document. He also showed that the Dynamic Help messages built automatically were not significantly less reliable and understandable than those written manually.7

Taken together, the research of Captains Wolven and Haines shows that on-line user support based on Dynamic Help is an attractive alternative to conventional help message techniques. The focus of their research (and the ISM project in general) has been on how to improve the user interface of existing software. The results of their research will allow the U.S. Army to provide a consistent and helpful user interface for many different software packages used today at installations around the world.

The final benefit of their work is to define a framework for the implementation of Dynamic Help in new software packages. If the pertinent information contained in the original requirements document is provided in a suitable format (data structure) then the components needed to provide Dynamic Help will be automatically available.
3. Common Practices in Requirements Documentation

Requirements documentation is part of an overall software engineering process. The requirements are "identifiable capabilities expressed as performance measures or functions that the system must possess to meet mission objectives." The requirements analysis phase of a project can be viewed as the problem definition, whereas the actual design of the system is the problem solution. Scharer (1981) identified several characteristics of a high-quality requirements definition:

1. strict separation of functional specification (what the system is to do) from system design (how the system is to work)
2. development of a definition that is ... easily translated into a physical system design
3. orderly decomposition of system requirements from the highest to the lowest level [top-down]
4. representation of system requirements as a logical model expressed in graphic terms, using a minimum of textual explanations.
5. separation of activities analysis from data analysis

Scharer also stresses the need to adopt a systematic approach to requirements documentation. The challenge in requirements documentation is to provide enough information to completely define the system (without designing it) and to present the requirements in an understandable format.

Requirements can be subdivided into functional and non-functional; both need to be well documented. Functional requirements detail what the system must accomplish. Non-functional
requirements identify constraints, limitations or expectations that need to be considered by the designer. Yeh and Ng (1990) state that "the most effective way to understand a complex system is to develop a model of the system. The model provides a basis for discussion among those concerned with the requirements derivation." This idea is well accepted; the only difference among authors is which models are best for different situations. In requirements documentation, models can be either conceptual or operational (as in the case of rapid prototyping).

Conceptual models improve our ability to understand a complex system. Coad and Yourdon (1990) describe four conceptual approaches to requirements analysis: functional decomposition, dataflow (structured) analysis, information modeling, and object-oriented analysis.

Functional decomposition represents the system in terms of functions, sub-functions and functional interfaces. The emphasis is placed on the processing that will be performed by the system.

Dataflow (or structured) analysis models identify all major system activities and then their associated inputs and outputs. Shamlin (1990) recommends a model of this type. Her "Input-Process-Output" model for requirements definition can be used to describe an entire system and any of its subsystems or functions. Shamlin further decomposes a computer system into the external, interface, and internal systems, and the data flows that communicate between them. Geoffrion (1987) develops a structured analysis framework which captures the semantic and mathematical relationships between
various systems elements on an acyclic graph. Elements in the system are classified as either "primitive" (requiring no other system elements for a complete definition), or those whose definition can only be written in terms of other elements in the system. Geoffrion's techniques not only produce a conceptual model of the system, but also require the modeler to define and catalog all parts of the system.15

Information models identify objects in the problem space and their associated attributes and relationships. Object-oriented analysis carries information modeling several steps further by adding processing requirements (services), a classification structure, and relationships between attributes and processes (inheritance).16

Operational models of a system give the user and the designer a chance to view a working model which incorporates the major ideas of the desired final product. Operational models can be either manual or automated; they can be used when requirements are expressed as behavioral requirements, by which the software designer is effectively told to design a system that 'behaves like this'. Behavioral requirements are expressed in a set of simulated interactive session records and simulated input and output files. The requirements author can demonstrate the operational model either by hand or by preparing a program using a specification language (e.g. APL) or a rapid prototyping tool software utility. In either case, the requirements may be accurately and comprehensively expressed without the need to give appropriate names to all variables and functions. However, the best practice would be to assign names to all
high-level variables and high-level functions at this stage, since these should not be affected by the software architecture and design.\textsuperscript{17}

Operational models lack the size, speed and finishing touches of a completed software product but the major concepts can be observed. Yeh and Ng (1990) identify three advantages of operational models:

The first advantage is that the rigor of having to make a model that 'runs' always proves to be a powerful influence against ambiguity and vagueness in requirements. The second advantage of operational specifications is that they provide natural structures to which performance requirements can be attached. ... A third advantage is that operational specifications make it possible to include resource requirements when necessary\textsuperscript{18}

Structural concepts which should be employed along with any requirements approach include simplification, abstraction, partition and projection.\textsuperscript{19}

4. Matching Dynamic Help to Requirements Documentation

The incorporation of Dynamic Help into a new software system reflects a strong interest in the quality of the user interface. Building a quality interface and an efficient system demands a thorough requirements analysis and subsequent documentation. The better the requirements analysis, the easier it is to incorporate Dynamic Help into the system. Systems developed primarily through operational modeling and then implemented directly through automated code generation are generally not good candidates for Dynamic Help because the requirements are never expressed in human language. Additionally, this approach indicates a concern for efficiency in developing an early version rather than for efficiency in using a mature
version. On the other hand, systems used hundreds of times daily by a myriad of users, like those systems in the ISM project, will probably deserve and receive more attention to the user interface, and thereby a more detailed requirements document. In these cases, the cost to provide a more detailed requirements document is amortized over thousands of users and a time span of many years.

Many approaches to requirements analysis and documentation are available. The best one for a particular problem is often a matter of great debate. Whatever method is used, it must eventually define the objects in the system and the functions the new system must perform. The documentation can be matched to the needs of Dynamic Help as soon as the requirements analyst or system designer specifies variable names and domains.

Dynamic Help can be incorporated into the system by three basic methods: 1) in-line help logic can be mixed in with the system code; 2) data can be passed to a Dynamic Help module where the messages can be assembled, or 3) a Dynamic Help module can be designed with the ability to issue queries of its own. Method #1 requires human intervention and decisions at many points in the system; method #2 requires human decisions in advance about what parameters need to be passed into the module. Both of these defeat the purpose of attempting to automate the Dynamic Help message building process. Only the third method is compatible with the automation of Dynamic Help.

Given that the Dynamic Help module must be able to issue queries about the state of the system and its parameters, there are various
mechanisms by which these queries can be made. If the new system is DBMS-based, a suitable query structure is readily available through statements in a query language (e.g. SQL) that can be written into the Dynamic Help module. If the new system is not DBMS-based, a suitable data structure and global variables will need to be created. In both cases however, it is highly likely that the system designer will have created these structures as part of the system development. Global variable or database tables will probably have been specified to create dictionaries of such things as screen (or window) names, variable names, modules, and high-level functions. All of these dictionaries contain items that can be used as parameters in Dynamic Help sentences.

A Dynamic Help "sentence" is made through the combination of run-time state variables and either stored or queried text. The eight standard Dynamic Help sentences are shown in Figure 1. Not every sentence is applicable every time Dynamic Help is invoked. At times one or more sentences could be null.

One of the goals of this paper was to develop a strategy which would lead to the automatic creation of material for Dynamic Help during the requirements analysis and design specification of a new system. If the high-level dictionaries discussed above have been created, then only small adjustments are necessary for the existing information to be compatible with Dynamic Help.

A complete Dynamic Help message consists of all of its sentences (including any null sentences). The variable phrases are materialized into text by query and assembled with the constant phrases to form
the single text object that is the message. Figure 1 shows the eight Dynamic Help sentences that will be discussed in detail below.

4.1 Context
The Context sentence orients the user. A set of relational dictionaries linking object, command and window dictionaries can provide Dynamic Help with the information it needs to write this sentence. The dictionaries must be written so that each user-task description is an imperative verb phrase. The phrase is a command: preceded by the verb auxiliary "to", it is an infinitive verb phrase. No other verb inflections are needed in Dynamic Help messages. An example would be: "Ready to print clothing record" (italics portion dynamically retrieved from function dictionary and concatenated to the standard prefix "Ready to. . .")

4.2 Navigation
There are both global and local navigation sentences. Global navigation occurs among screens, windows, and menus. A dictionary of screens, window names, and menu names should be constructed which lists them in their ordinary logical sequence. The dictionary should also contain the names of the keyboard keys which will cause the cursor to move to the previous or next screen, window, or menu. Local navigation is handled dynamically only by exception -- most of the local navigation instructions are treated in the General sentence. A local navigation sentence can be built dynamically if an exceptional circumstance warrants its use. (e.g. a special interactive
procedure used only in the currently active window, as for scheduling, partitioning, rearranging or assembling objects).

4.3 Meaning

The meaning sentence is a short semantic definition of the datum currently being entered. When required, it can be extracted from the appropriate dictionary (i.e. the meaning of an object is retrieved from the Object dictionary). The meaning of a function should be written as a third person singular verb phrase followed by an object or predicate (subject omitted). The meaning of a variable or object should be written as a noun phrase.

4.4 Domain

The domain sentence describes the acceptable entries for the current field when Dynamic Help is invoked. This sentence can be thought of as a constraint definition of the acceptable entries. The information for this sentence can be written as a part of the global variable dictionary in several sections: type (e.g. character or number), range and acceptable values (e.g. positive or negative). An example: "Acceptable entries are zero and positive numbers" (italics dynamically retrieved from global variable dictionary).

4.5 Content

The content sentence provides the user with a list of choices or entries that are legal given the context and state. This information can be stored in the Global Variable dictionary or the Commands dictionary, whichever is more appropriate. Recognition is easier than
recall and this sentence gives the user a list from which to choose an entry.

4.6 Format

The format sentence provides an example of a correct entry or provides a short description of the correct format, whichever is more informative. Sometimes the context and format sentences can be expressed as a single statement or list. If the format sentence gives examples, a representative sample can be retrieved from previously entered data. The example system in Section 5 is largely window-based and the Format sentence is used to communicate an input protocol based on the user's context.

4.7 General

The last sentence in the Dynamic Help message is the General sentence. This sentence tells the user how to retrieve context-independent documentation which explains concepts and protocols which are uniform throughout the system. This includes local navigation protocols governing window management, cursor control, scrolling instructions, display control; confirmation, undo, cancel, and back protocols; and the basic effects of particular keys (e.g. return, backspace, function, and escape keys). This General documentation will be a static text file which is retrieved only when specifically requested by the user; the text file is not retrieved every time Dynamic Help is invoked. The General sentence in the Dynamic Help message tells how to retrieve it.
The dictionaries called for by the eight above sentence descriptions are summarized below:

- Global variables
- High-level functions
- Objects
- Screens (or windows)
- Commands
- General Documentation

Once these dictionaries are in place, in a form that can be queried, with appropriate text that makes sense, then they can be queried not only by the Dynamic Help module, but also by the system's designer and programmer. The same sentences that are built and accessed by Dynamic Help can be queried to automatically write descriptive subroutine headers, user manuals, or other documentation. A very large part of the system's entire documentation needs can thus be supplied in a highly accessible form near the beginning of the development process.
5. Demonstration of Universal Documentation

5.1 Illustrative Example: University Registration System

To demonstrate the use of Dynamic Help as an aid to both documentation and user interface, we will use the familiar setting of a university-level computerized student registration system. A functional outline for the simulated system is shown below:

<table>
<thead>
<tr>
<th>Log-on Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verify student ID number</td>
</tr>
<tr>
<td>Retrieve student data</td>
</tr>
<tr>
<td>Retrieve course data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Registration functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register students for classes</td>
</tr>
<tr>
<td>Print a student's schedule</td>
</tr>
<tr>
<td>Display course offerings</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Records Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display grade reports</td>
</tr>
<tr>
<td>Process transcript requests</td>
</tr>
<tr>
<td>Process change of address requests</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Financial Aid Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide general information</td>
</tr>
<tr>
<td>Process applications for financial aid</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Housing Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process applications for campus housing</td>
</tr>
<tr>
<td>Process room change requests</td>
</tr>
</tbody>
</table>

The scope of this example is limited to the user interface requirements for the Registration portion of the system with particular emphasis on the “Register students for classes” function: other menu items and the system administrator functions, such as
course availability updating are not covered. Viewed in greater detail

the Registration Functions are as follows:

Registration

- Register students for classes
- Declare primary courses
- Declare alternate courses
- Interchange primary and alternate courses
- Registration closure (request official acceptance of a
  schedule)
- Print a student’s schedule
- Display course offerings

The General documentation sentence tells how to get information

on the following functions:

<table>
<thead>
<tr>
<th>Non-Functional Requirements</th>
<th>Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Touch input</td>
<td>Click mouse or use light pen on displayed command button, or move cursor to button and press return</td>
</tr>
<tr>
<td>2. Typed input</td>
<td>Type a brief command code and press return</td>
</tr>
<tr>
<td>3. Undo</td>
<td>Return to the state immediately preceding the last major action; a second invocation restores the undone result</td>
</tr>
<tr>
<td>4. Window management</td>
<td>Click mouse to make a window active; windows can be moved or re-sized</td>
</tr>
<tr>
<td>5. Print output</td>
<td>requested through touch or typed commands</td>
</tr>
</tbody>
</table>

Dictionaries were written to define Global variables. High-level
functions, Objects, Windows, Command, and Input protocols. Several
tables relating the various dictionaries were constructed that further
define the system and make it possible for the Dynamic Help module to construct its messages.

As discussed in Section 4, we are assuming that the Dynamic Help module has the ability to issue queries to these dictionaries and can then build the help messages. The Dynamic Help module can supply the proper punctuation, conjunctions and indefinite articles (an) when the messages are assembled. These auxiliary functions reside entirely within the Dynamic Help module and are not a part of the system's functional code.

The auxiliary functions of the Dynamic Help module include:

<table>
<thead>
<tr>
<th>Auxiliary Function</th>
<th>Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Indefinite article generation</td>
<td>Change the indefinite article “a” to “an” when followed by a string whose first two characters form a vowel sound or when the first word is listed in an exceptions dictionary.</td>
</tr>
<tr>
<td>2. List assembly</td>
<td>When several items of equal status are retrieved, separate them by commas and insert “and” before the last item; if the list is long or has an internal structure, then build a table.</td>
</tr>
<tr>
<td>3. Sentence punctuation</td>
<td>Capitalize the first letter of each sentence, separate clauses with commas (or semicolons) and put a period at the end of the sentence.</td>
</tr>
</tbody>
</table>

Whenever Dynamic Help is invoked, the Dynamic Help module views the user's current situation in terms of what part of the system are "active". The following Dynamic Help protocol is established for this example (in order of search priority):
Context Search Protocol

1. Query for an active command and object
2. Query for an active object in an active window
3. Query for an active window
4. Query for an active menu item

As soon as a search is successful the Dynamic Help module queries the appropriate tables and dictionaries based on the user's situation and then assembles the help message.

There are six windows used to register for courses: Department, Course Offerings, Schedule, Command, Prompt, and Dynamic Help. Their respective definitions are given in the Window Dictionary. Each window has a title bar and scrolling capability. The Dynamic Help window forms only when Dynamic Help is invoked and is discarded by the user after being read. The command and prompt windows are always active and the other windows can be made active by clicking on the title bar. Non-active windows are visible but subdued. The command window consists of command buttons which are subdued whenever that particular command is unavailable. The Prompt window provides the user with prompts as shown in the following table:
Prompt Message | Meaning
---|---
Wait | System is processing and is not ready for input
Touch | System is ready for only touch input (e.g. mouse, arrow key, light pen)
Type | System is ready for only typed keyboard input
Touch or Type | System is ready for either touch or typed input
Retouch | System is ready for input; last input was touch and was not accepted
Retype | System is ready for input; last input was typed and was not accepted
Confirm | (When Confirm protocol is to repeat input)
Y or N | (When Confirm protocol is to answer yes/no question)

Figure 3. Example Screen for the University Registration System
5.2 Registration Function

The Registration function brings together a student and a particular course (or set of courses) to form a registered schedule. The student builds his/her proposed schedule by selecting a department, then a course offered by that department, and then by designating a course as either Primary or Alternate. As courses are designated they appear in the Schedule window. A primary course appears on the schedule in normal type; an alternate course appears in subdued type. The following protocols are used for building a schedule:

1. At most, two instances (sections) of a course can be on the screen at any time, one primary and one alternate

2. If a third instance of a course is added as primary, the current primary becomes the new alternate and the old alternate is removed from the schedule completely

3. If a third instance of a course is added as alternate, it takes the place of the current alternate and the old alternate is discarded automatically

4. If an alternate course on the schedule is designated as primary, the primary instance of that course (if any) becomes the new alternate

5. The command to Remove a course deletes all instances of the course from the schedule, whether primary or alternate

6. Only one instance of a course (primary or alternate) can occupy a single cell

When the Schedule window is active, an "Accept Schedule" button is active in the Command window. By clicking this button the
student will register for all primary (non-subdued) classes shown in the schedule window.

<table>
<thead>
<tr>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday</td>
</tr>
<tr>
<td>0800</td>
</tr>
<tr>
<td>0900</td>
</tr>
<tr>
<td>1000</td>
</tr>
<tr>
<td>1100</td>
</tr>
<tr>
<td>1200</td>
</tr>
<tr>
<td>1300</td>
</tr>
<tr>
<td>1400</td>
</tr>
<tr>
<td>1500</td>
</tr>
<tr>
<td>1600</td>
</tr>
<tr>
<td>1700</td>
</tr>
<tr>
<td>1800</td>
</tr>
</tbody>
</table>

Figure 4. Example Schedule Window
(Italics used to indicate subdued print)

A typical user session (in the context of this example) is as follows:

1) The student logs on to the system using his/her nine-digit student ID number as identification.

2) The student chooses, in order, the "Registration" and the "Register for Classes" menu items. The Command, Prompt, Department, Course Offerings, and Schedule windows form automatically. The student then invokes Dynamic Help which builds a run-time help message (see message #1).

3) The student clicks the department window title bar to make it the active window. He then highlights the department name Industrial Engineering in the window and presses return. The Course Offerings window is made active and brought to the front.

4) The student finds ISYE6650A1 in the Course Offerings window, highlights it and then invokes Dynamic Help (see message #2).
5) The student then activates the “Primary” command button and invokes Dynamic Help (see message #3). After reading and then disposing of the Dynamic Help message, the student presses return (combining the course object and the command) to add the course to his proposed schedule. Next, the student highlights ISYE6650A2 (a different section) in the Course Offerings window and clicks the “Alternate” button to add a second section of the same course to his proposed schedule.

6) The student then activates the Schedule window (no objects or commands are active). ISYE6650A1 and ISYE6650A2 are present on the schedule because they were added in the previous step. The student invokes Dynamic Help and message #4 is built.

7) The student highlights the ISYE6650A1 object on the schedule and then invokes Dynamic Help (see message #5).

8) The student then activates the “Accept Schedule” command and invokes Dynamic Help resulting in message #6. A press of the return key registers the student’s schedule.
Help Message #1

User Situation

Active Menu Item: Register for Classes
Active Window: None
Active Object: None
Active Command: None

Context: Ready to <create or modify your academic class schedule>.

Meaning: <null>

Domain: <null>

Content: <null>

Format: <Activate a window by clicking on its title bar>.

Navigation: To return to <the registration menu><press the escape key>.

General: To see general documentation about this system press F1.

The message as it would appear to the user:

Ready to create or modify your academic class schedule. Activate a window by clicking on its title bar. To return to the Registration Menu, press the escape key.

To see general documentation about this system press F1.
Help Message #2

User Situation

Active Menu Item: Register for Classes
Active Window: Course Offerings
Active Object: IYSE6650A1
Active Command: None

Context: Ready to <choose a command for> <ISYE6650A1>.

Meaning: <Null>

Domain: Acceptable commands are <P, A, and T>.

Content: The commands mean <(P)rimary, (A)lternate, and (T)ime remaining>.

Format: <Enter a command and press return to apply it to the active item>.

Navigation: To return to <the registration menu> <press the escape key>.

General: To see general documentation about this system press F1.

The message as it would appear to the user:

Ready to choose a command for ISYE6650A1. Acceptable commands are P, A, and T. The commands mean (P)rimary, (A)lternate, and (T)ime remaining. To return to the registration menu press the escape key.

To see general documentation about this system press F1.
Help Message #3

User Situation

Active Menu Item: Register for Classes
Active Window: Course Offerings
Active Object: IYSE665OA1
Active Command: P

Context: Ready to <add><ISYE6650A1><as a Primary course on your schedule>.

Meaning: <Null>

Domain: Acceptable commands are <P, A, and T>.

Content: The commands mean <(P)rimary, (A)lternate, and (T)ime remaining>.

Format: <Press return to apply the command to the highlighted item>.

Navigation: To return to <the registration menu ><press the escape key>.

General: To see general documentation about this system press F1.

The message as it would appear to the user:

Ready to add ISYE6650A1 as a Primary course on your schedule. Acceptable commands are P, A, and T. The commands mean (P)rimary, (A)lternate, and (T)ime remaining. Press return to apply the command to the highlighted item. To return to the registration menu press the escape key.

To see general documentation about this system press F1.
Help Message #4

User Situation

Active Menu Item: Register for Classes
Active Window: Schedule
Active Object: None
Active Command: None

Context: Ready to <modify or finalize your course schedule>.
Meaning: <Null>
Content: The commands mean <(R)emove, (S)earch, (D)isplay (P)rimary, (D)isplay (A)lternate, (C)hange (B)asis, and (T)ime remaining>.
Format: <Activate an item by highlighting; press return to select it>.
Navigation: To return to <the registration menu><press the escape key>.
General: To see general documentation about this system press F1.

The message as it would appear to the user:

Ready to modify or finalize your course schedule. Acceptable commands are R, S, DP, DA, CB, and T. The commands mean (R)emove, (S)earch, (D)isplay (P)rimary, (D)isplay (A)lternate, (C)hange (B)asis, and (T)ime remaining. Activate an item by highlighting; press return to select it. To return to the registration menu press the escape key.

To see general documentation about this system press F1.
Help Message #5

User Situation

Active Menu Item: Register for Classes
Active Window: Schedule
Active Object: ISYE665OA1
Active Command: None

Context: Ready to <choose a command for >ISYE665OA1>.  
Meaning: <Null>
Content: The commands mean <(R)emove, (S)earch, (C)hange (B)asis (D)isplay (P)rimary, (D)isplay (A)lternate, and (T)ime remaining>.  
Format: <Enter a command and press return to apply it to the active item>.  
Navigation: To return to <the registration menu><press the escape key>.  
General: To see general documentation about this system press F1.

The message as it would appear to the user:

Ready to choose a command for ISYE665OA1. Acceptable commands are R, S, CB, DP, DA, and T. The commands mean (R)emove, (S)earch, (C)hange (B)asis (D)isplay (P)rimary, (D)isplay (A)lternate, and (T)ime remaining. Enter a command and press return to apply it to the active item. To return to the registration menu><press the escape key.

To see general documentation about this system press F1.
Help Message #6

User Situation

Active Menu Item: Register for Classes  
Active Window: Schedule  
Active Object: ISYE6650A1  
Active Command: R

Context: Ready to *register for the primary course(s) in shown in the Schedule window*.

Meaning: *<Null>*


Content: The commands mean *(R)emove, (S)earch, (C)hange (B)asis (D)isplay (P)rimary, (D)isplay (A)lternate, and (T)ime remaining*.

Format: *<Press return to apply the command to the highlighted item>*.

Navigation: To return to *<the registration menu *> <press the escape key>*.

General: To see general documentation about this system press *F1*.

The message as it would appear to the user:

Ready to remove ISYE6650A1 from your schedule. Acceptable commands are R, S, CB, DP, DA, and T. The commands mean (R)emove, (S)earch, (C)hange (B)asis (D)isplay (P)rimary, (D)isplay (A)lternate, and (T)ime remaining. Press return to apply the command to the highlighted item. To return to the registration menu press the escape key.

To see general documentation about this system press F1.
Dynamic Help Dictionaries

### Global Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
<th>Type</th>
<th>Range</th>
<th>Acceptable Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>term</td>
<td>identifies the academic quarter being considered</td>
<td>character</td>
<td>A-Z</td>
<td>summer, fall, winter, spring</td>
</tr>
<tr>
<td>ID number</td>
<td>uniquely identifies each student</td>
<td>integer</td>
<td>1-99999999999</td>
<td>positive, nine-digit number</td>
</tr>
<tr>
<td>section number</td>
<td>uniquely identifies each course offering</td>
<td>integer</td>
<td>1-10000</td>
<td>positive integer</td>
</tr>
</tbody>
</table>

### High Level Functions

<table>
<thead>
<tr>
<th>function</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>register a student</td>
<td>match student and courses to form a registered schedule</td>
</tr>
<tr>
<td>print a schedule</td>
<td>print an existing or proposed schedule</td>
</tr>
<tr>
<td>display course offerings</td>
<td>display courses offered for the current term</td>
</tr>
</tbody>
</table>

### Objects

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>registration</td>
<td>consists of a student ID number and a course serial number</td>
</tr>
<tr>
<td>student</td>
<td>consists of an ID number, classification, academic standing</td>
</tr>
<tr>
<td>course</td>
<td>consists of the section serial number, department, class number, term and a professor</td>
</tr>
<tr>
<td>time slot</td>
<td>consists of the day of the week and the hour of the class</td>
</tr>
<tr>
<td>location</td>
<td>consists of the building number and the zone on campus</td>
</tr>
</tbody>
</table>
## Windows

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>department</td>
<td>display all academic departments</td>
</tr>
<tr>
<td>course offerings</td>
<td>display all classes offered by the selected department</td>
</tr>
<tr>
<td>schedule</td>
<td>display your proposed course schedule.</td>
</tr>
<tr>
<td>command</td>
<td>allow the entry of commands</td>
</tr>
<tr>
<td>prompt</td>
<td>provide system information</td>
</tr>
</tbody>
</table>

## Commands

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>(P)rimary course</td>
<td>add a primary course to the schedule</td>
</tr>
<tr>
<td>A</td>
<td>(A)ltimate course</td>
<td>add an alternate course to the schedule</td>
</tr>
<tr>
<td>R</td>
<td>(R)emove a course</td>
<td>remove a course from the schedule</td>
</tr>
<tr>
<td>S</td>
<td>(S)earch for a course</td>
<td>search for a course to fill the active time slot</td>
</tr>
<tr>
<td>CB</td>
<td>(C)hange (B)asis</td>
<td>change the basis of a course</td>
</tr>
<tr>
<td>DP</td>
<td>(D)isplay (P)rimary schedule</td>
<td>display your primary schedule</td>
</tr>
<tr>
<td>DA</td>
<td>(D)isplay (A)ltimate schedule</td>
<td>display your alternate schedule</td>
</tr>
<tr>
<td>T</td>
<td>(T)ime remaining</td>
<td>display time remaining in this session</td>
</tr>
<tr>
<td>AS</td>
<td>(A)cept (S)chedule</td>
<td>register for primary course (s) shown in the Schedule window</td>
</tr>
</tbody>
</table>
### Input Protocol

<table>
<thead>
<tr>
<th>Situation</th>
<th>Format Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>nothing active</td>
<td>Activate a window by clicking the mouse on its title bar</td>
</tr>
<tr>
<td>active window only</td>
<td>Activate an item by highlighting; press return to select it</td>
</tr>
<tr>
<td>active window and active object</td>
<td>Enter a command and press return to apply it to the active item</td>
</tr>
<tr>
<td>active object and active command</td>
<td>Press return to apply the command to the highlighted item</td>
</tr>
</tbody>
</table>

### Command and Windows Relational Table

<table>
<thead>
<tr>
<th>Commands</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>course offerings</td>
</tr>
<tr>
<td>A</td>
<td>course offerings</td>
</tr>
<tr>
<td>R</td>
<td>schedule</td>
</tr>
<tr>
<td>S</td>
<td>schedule</td>
</tr>
<tr>
<td>CB</td>
<td>schedule</td>
</tr>
<tr>
<td>DP</td>
<td>schedule</td>
</tr>
<tr>
<td>DA</td>
<td>schedule</td>
</tr>
<tr>
<td>T</td>
<td>department</td>
</tr>
<tr>
<td>T</td>
<td>course offerings</td>
</tr>
<tr>
<td>T</td>
<td>schedule</td>
</tr>
<tr>
<td>AS</td>
<td>schedule</td>
</tr>
</tbody>
</table>
### "Ready to" Relational Table

<table>
<thead>
<tr>
<th>Active Object</th>
<th>Active Command</th>
<th>&quot;Ready to&quot; suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>course</td>
<td>P</td>
<td>add &lt;course&gt; as a Primary course on your schedule</td>
</tr>
<tr>
<td>course</td>
<td>A</td>
<td>add &lt;course&gt; as an Alternate course on your schedule</td>
</tr>
<tr>
<td>course</td>
<td>R</td>
<td>remove &lt;course&gt; from your schedule</td>
</tr>
<tr>
<td>time slot</td>
<td>S</td>
<td>search for a course to fill the active time slot</td>
</tr>
<tr>
<td>course</td>
<td>CB</td>
<td>change the basis of &lt;course&gt;</td>
</tr>
<tr>
<td>course</td>
<td>DP</td>
<td>display your primary schedule</td>
</tr>
<tr>
<td>course</td>
<td>DA</td>
<td>display your alternate schedule</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Active Object</th>
<th>Active Window</th>
<th>&quot;Ready to&quot; suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>course</td>
<td>course offerings</td>
<td>choose a command for &lt;course&gt;</td>
</tr>
<tr>
<td>course</td>
<td>schedule</td>
<td>choose a command for &lt;course&gt;</td>
</tr>
<tr>
<td>time slot</td>
<td>schedule</td>
<td>choose a command for &lt;time slot&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>active Menu Item</th>
<th>Active Window</th>
<th>&quot;Ready to&quot; suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>register...</td>
<td>department</td>
<td>choose a department</td>
</tr>
<tr>
<td>register...</td>
<td>course offerings</td>
<td>add courses to your schedule</td>
</tr>
<tr>
<td>register...</td>
<td>schedule</td>
<td>modify or finalize your course schedule</td>
</tr>
<tr>
<td>register...</td>
<td>command</td>
<td>select a command or enter a command at the prompt</td>
</tr>
<tr>
<td>print...</td>
<td>schedule</td>
<td>print a copy of your schedule</td>
</tr>
<tr>
<td>display...</td>
<td>department</td>
<td>choose a department</td>
</tr>
<tr>
<td>display...</td>
<td>course offerings</td>
<td>view courses from &lt;department&gt;</td>
</tr>
<tr>
<td>Active Menu Item</td>
<td>&quot;Ready to&quot; suffix</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>register for classes</td>
<td>create or modify your academic schedule</td>
<td></td>
</tr>
<tr>
<td>print your schedule</td>
<td>print a copy of your schedule</td>
<td></td>
</tr>
<tr>
<td>display course offerings</td>
<td>display course offerings for the current quarter</td>
<td></td>
</tr>
</tbody>
</table>
ENDNOTES


2Donovan Young, “Embedded User Support for U.S. Army Installation Software” 1990 [photocopy], p.3, School of Industrial and Systems Engineering, Georgia Institute of Technology.


4Young, "Embedded User Support", 1.


13Ibid, 277.
14 Shamlin, The Other Side of Software, 25.


16 Coad and Yourdon, "Object-Oriented Analysis", 279.

17 Donovan Young, Class notes, ISyE 6841: Decision Support Systems. (School of Industrial and Systems Engineering, Georgia Institute of Technology, 1988).

18 Yeh and Ng, "Software Requirements", 456.

19 Yeh and Ng, "Software Requirements", 453.

20 Coad and Yourdan, "Object-Oriented Analysis", 277.


22 Young, "Specification of User Requirements", 5.
BIBLIOGRAPHY


Smith, Christopher, Mike McCracken, Donovan Young. "Embedded User Support Project Plan", 1990 [photocopy]. School of Industrial and Systems Engineering, Georgia Institute of Technology.


