

**An Integrated Taxonomy
of On-Line Help Based on
User Interface View**

by

Piyawadee "Noi" Sukaviriya

**GIT-GVU-91-20
October 1991**

**Graphics, Visualization & Usability
Center**

**Georgia Institute of Technology
Atlanta GA 30332-0280**

AN INTEGRATED TAXONOMY OF ON-LINE HELP BASED ON USER INTERFACE VIEW

Piyawadee "Noi" Sukaviriya

College of Computing
Georgia Institute of Technology
801 Atlantic Dr., Atlanta, GA 30332-0280
Phone: (404) 894-9105
E-mail: noi@cc.gatech.edu

ABSTRACT

We develop a comprehensive help taxonomy by combining both user interface and help system attributes, ranging from help access interface, presentation, and supporting knowledge structure, to implementation. The taxonomy systematically identifies independent axes along which help can be categorized which in turn encloses a space of help categories in which to place currently existing help research, and identifies distinct help software architectural features which contrast pros and cons in different approaches to implement help systems. The taxonomy projects a vision of what help can be like if it is on a par with advances in user interface technology, and desirable design features of help system architectures which are in the progressive direction along with the user interface software tools.

KEYWORDS: Help, Help Taxonomy, Help Architecture, User Interface Environment

INTRODUCTION

The problem of producing quality on-line help is addressed in a number of research areas and is characterized, albeit sometimes incorrectly, by both those who use it and those who build it. Although help is one of the more important components of a user interface, previously the attention it receives from those researchers who focus on a total system or a system design model is minimal, suggesting that help plays a minor rather than a major role. It has been difficult to develop well-focused help research agendas based on previous results, since those results were very often colored and shadowed by research objectives of which help was simply a by-product. The taxonomy implicitly suggests that help tool design could and might best progress along with the user interface state-of-the-art technology to ensure its compatibility and effective, direct communications.

A major problem in defining an appropriate help research agenda has obviously been the lack of integration of the different research contributions necessary to establish both a direction and some historical foundation for the research. Other problems include the lack of an appropriate taxonomy

to define help information content, to model access to help methods, and to categorize strategies used to generate help. In this paper, we establish a help taxonomy by combining both user interface and help system attributes, ranging from help access interface, presentation, and supporting knowledge structure, to implementation.

OVERVIEW OF TAXONOMY

There have been a few efforts to taxonomize help characteristics. Borenstein categorized the access and presentation aspects of help [Borenstein85]; access is differentiated by whether the system or the user initiates help, and user-initiated help is further differentiated by the means (keyword access, menu, spoken words, etc.) by which access is generated. Presentation is differentiated by the method used to present help (scrolling text, window, speech, pictures, etc.) and by the information source for the presentations (on-line manual text or representations). [Kearsley88] and [Balzert87] also taxonomized system-initiated and user-initiated help access.

In addition to the initiating agent of help, both Kearsley and Balzert described static and dynamic features of help. Static help presents pre-stored information without modifications; dynamic help refers to help content which varies according to runtime context and/or individual user preferences. Kearsley explicitly considered systems which, though only presenting pre-stored information, provide somewhat intelligent help access mechanisms to present different pieces of help text based on recent history of user actions as dynamic help.

Lutze [Lutze87] discriminated help features based on the kind of service delivered and stages of performing computer tasks; the most important stage is the executing action stage. Lutze further categorized the help provided in this stage as one of two kinds: one responding to *how* questions, the other to *why* questions.

The researchers referred to above provide backbone taxonomies for systematically characterizing help systems. However, they are neither discrete nor broad enough to describe the architectural aspects of help systems. Borenstein made only a simple distinction in this direction between using manual text and representations as an information source. Lutze modeled help categories from the service and the context points of view but did not consider

help system architecture as part of his taxonomy. Kearsley did not address architectural issues, and Balzert & Lutze proposed only generally that help architecture should be integrated as part of an application system. Technology on user interface software tools has progressed tremendously and if help is to be an integral part of these tools, characterizing the architecture for integration belongs in any taxonomy of help systems which is to capture and advance useful models of help systems. The following table suggests a highest level set of categories for describing help systems.

Table 1 suggests a comprehensive view which separates help interfaces from their underlying internal structures and presents the two highest categories of help components as help interface components and help system strategy components. The interface view characterizes help based on what the user perceives of help systems, which is characterized further into how to access help and help presentation. Help presentation includes both the content substance and how it is presented. The system strategy

Help Interface (User View)	Access
	Content
Help System Strategy (System View)	Information Structure
	Architecture

Table 1 An Overview of Help Taxonomy

view is from the implementation point of view. Ideally, system strategies should be independent of the help interfaces. However, limitations on some strategies certainly impose limitations on the interfaces a help system can provide. The following two sections elaborate the interface and the system strategy views in more detail.

Help Interface: the User View

Similar to Borenstein's taxonomy, two aspects of help access are summarized in Table 2: who initiates help, and how access is achieved when the user is the initiator. In addition, three categories in help presentation are summarized—the nature of help content, the medium and presentation style of help, and the level of help's adaptiveness to its environment. This table establishes the context in which previous help research will be placed, as will be discussed in the following five sections.

Passive vs. Active Help

System-initiated help monitors user activities and intervenes when the user makes mistakes [Kearsley88] or diverges from known goals. This kind of help is often

referred to as *active* help [Fischer85, Hecking87]. Genesereth incorporated knowledge of user courses of actions in Lisp procedures [Genesereth82] and used it to recognize a student's misunderstandings in mathematical reasoning tasks. Fischer stored knowledge about the user's conceptual understanding of the system, individual sets of tasks, specific tasks already performed, and advice already given to the user to help decide when the help system should intervene [Fischer85]. Hecking used stored optimal paths to recognize when the user did not utilize such paths to achieve tasks [Hecking87]. System-initiated help is always alert and obviously requires extensive knowledge of expected user behaviors and goals in the particular application domain it supports in order to be active.

User-initiated help, on the other hand, does not monitor the user and hence requires less information storage. It requires the user to make explicit requests to access help information. In contrast to the term active help, user-initiated help is also called *passive* help [Fischer85, Hecking87]. Traditional help systems are passive. The help research prototyped and developed by researchers mentioned in the previous paragraph, while emphasizing the active capability of help, were built to provide passive help as well.

Forms of Help Request

Passive help requires that users ask questions when they have problems. In traditional textual interfaces, explicit help requests may be posed by having the user form *textual* questions, either using a set of limited *keywords* known to help subsystems or using *natural language* such as in the Unix Consultant (UC) system [Wilensky84, Chin86].

With interfaces advancing and the use of multiple media becoming common, explicit textual requests may use or be combined with other media such as *graphical*, *gestural*, or *spoken* forms, to better convey the intention of user requests. It is potentially more natural when the forms used to access help are consistent with the interface styles being used. Graphical help access ranges from minimally graphic, such as selecting a help topic from a menu as normally supported in the Macintosh interface, to highly graphical such as pointing at graphical objects on the screen to form help questions. VSTAT's help [Stevens87] is an example of the latter where the user can combine textual questions with pointing at objects on the screen.

Procedural vs. Conceptual Help

Breuker identified two kinds of knowledge important for a user to acquire skills: support and operational knowledge [Breuker87]. *Support knowledge* gives context-independent meaning which justifies the operations used; the operations are chosen by the user based on her *operational knowledge* which is used for solving task-related problems. Barnard mentioned the need for *procedural knowledge* to support user translation of cognitive tasks – understandings of what a computer application program can do for them – into actions, which call for concepts of actions and their

parameters to execute the tasks [Barnard87]. Gwei used an educational approach to prescribe a good help system as being equipped by *interpretive* (what) and *descriptive* (how) explanations [Gwei90]. Carroll and Aaronson conducted a study to understand the importance of conceptual "how-it-works" and procedural "how-to-do-it" help separately in user learning to perform tasks [Carroll88]. Despite the different

The two kinds of help which provide the two kinds of support, about concepts and procedures, will be termed *conceptual* help and *procedural* help in our taxonomy. For the sake of simplicity, other types of help, such as help on errors or help on WHY questions, are subsumed in the conceptual help category. In fact, Bieger and Glock further broke down information needed in written instructions for procedural tasks into six other types [Bieger87]; we make a referenced to their work here but will subsume their categories in our conceptual category.

Textual, Graphical, and Other Forms of Help

Possibilities for help presentation have followed advances in user interfaces. The traditional text-oriented interface for decades has dictated text as the dominant presentation form of help. With the advent of graphical interfaces, presentation of help was no longer restricted to textual format. Static pictures have been used in help [OPENLOOK90] to relate explanations of concepts with what the user sees in graphical interfaces. However, static pictures do not achieve the full-fledged visualization, which other presentation forms could achieve, of the overall dynamic procedures involved in advanced interfaces.

As higher-performance computers and higher-resolution screens become common, aesthetic graphical animation is no longer a luxury. A few attempts to use graphical animation in help marked early innovations in this direction. Cullingford used pre-stored animated demonstrations to support context-sensitive textual help

terminologies used by these researchers, they agree on the distinction between knowledge of task concepts and of task procedures, and that they complement each other. Gong made explicit that conceptual assistance is especially useful when the application of previously learned procedures is required in new situations [Gong90].

explanations in his CADHELP system [Cullingford82]. Neiman extends to CADHELP a generation of graphical animation from the scriptal knowledge originally used for textual explanations [Neiman82]. Feiner, though he did not use graphical animation, used graphical illustrations indicating movements in his automated explanation system, APEX [Feiner85]. We developed an earlier prototype system to demonstrate examples of context-sensitive animated help [Sukaviriya88] which used simple two-dimensional animation of input devices to illustrate interactive procedures in an application context. An automatic generation of it was developed later as a system called Cartoonist [Sukaviriya91]. Graphical animation is another dimension in presenting help. Other media, such as speech audio, are conceivably becoming suitable in various situations. Robinson has concluded that whether information should be presented textually, verbally, or graphically must be determined by the kind of tasks users have to perform or responses they have to give back to the system [Robinson87].

In Table 2, five different forms of presentations hence are summarized—*text, static graphics, animation, audio, and multimedia*. The latter categorizes help presented in a combination of media, such as COMET [Feiner90], a presentation style being seriously studied and used to utilize the growing interactive multimedia technology and the combined effectiveness of verbal and visual information.

Help Interface (User View)	Help Access	INITIATOR: System-initiated/User-initiated
		ACCESS METHOD: Keyword-based/Natural-language-based/ Graphical/Gestural/Spoken
	Help Presentation	CONTENT: Conceptual/Procedural
		MEDIUM: Textual/Graphical/Animated/ Audio/Multimedia
		ADAPTIVENESS: Static/Context-sensitive/ History-sensitive/User-sensitive

Table 2 Help Interface Characteristics

Help System Strategy (System View)	Information Structure	Declarative Knowledge	Linear Text
			Entity-based
		Procedural Knowledge	Rule-based
			Grammar-based
			State-based
	Architecture	INTEGRATION: Isolated Help Module/Integrated with UI Tool	
		DESIGN-TIME KNOWLEDGE SHARING: Separated Help Knowledge/ Shared Help Knowledge with UI Tool	
		DOMAIN DEPENDENCE: Application-dependent/Application-independent	

Table 2.3 Help System Strategy Characteristics

Static vs. Sensitive Help

Static help uses pre-stored help information; generating it at runtime is a matter of accessing and presenting the information as stored. By this definition, commercial help systems such as Unix™ man pages, Microsoft Word help [Microsoft89] and OPEN LOOK spot help [OPENLOOK90] are considered static in our taxonomy. *Sensitive* help, by contrast, is generated directly from representations or data structures, and can be made appropriate to user needs by using various contextual factors to vary explanations, either by selectively retrieving appropriate information tokens or selecting different generation strategies. Generating help on the fly at runtime hence allows for incorporating sensitivity to help content, the kind of help referred to as *dynamic* in [Lutze87, Balzert87, Kearsley88].

Dynamic help can be sub-categorized to a finer grain such as that suggested by other categories listed in Table 2. Help is referred to here as *context-sensitive* if its content varies depending on the current runtime context such as program state, screen state and data state, and includes currently existing application entities in its explanations. The latter includes trivial sensitivity such as in the *File 'xxx' not found* message where 'xxx' is always replaced by

Help Strategies: the System View

One type of knowledge Carroll and McKendree concluded that a help system needs to have in order to be intelligent is

the name of the file in question. Context-sensitive help in our taxonomy also includes help which varies the length of help presentations when the current context is not appropriate and it is necessary to add additional components to set up a "help stage". Lutze called this kind of context-sensitivity *projective*, which refers specifically to providing explanations of how to achieve the user's desired goals from the current state [Lutze87].

Help content can also vary according to user history, for example, the recent commands, or certain keywords selected by the user, interaction techniques chosen, input devices used, how the user got to the current state, and so forth. This kind of help is referred to as *history-sensitive*. An example is the help system reported in [Senay87] which provided help based on the user's history of frequency of command usage. Another category of "sensitivity" is help which varies the level of explanations for individual users based on how knowledgeable they are, their preferences, their accumulated efficiency with different system tools and resources, etc.; such help is termed *user-sensitive*. [Travellyan87, Chin86] reported help systems in this category. History-sensitive and user-sensitive help, shown as distinct categories in Table 2, are grouped under the one category of *adaptive* help in [Lutze87, Balzert87].

(application) domain knowledge [Carroll87]. Table 3 portrays two major types of domain knowledge representation—declarative or procedural—and we will discuss

how help generation mechanism is structured to utilize the domain knowledge. Independent of knowledge structures, architectural characteristics presented in Table 3 are used to classify systems on the basis of how much runtime context and how much design-time knowledge help systems share with user interface software, and how dependent the architecture and the knowledge structure are on the application domain being supported. Existence of information sharing suggests the degree of sensitivity which could be incorporated into help explanations, and degree of domain independence indicates how portable the help systems are.

With Table 3 as a framework, the following three subsections discuss previous work categorized using one or more of these "system" aspects of help system strategies. The first two sections discuss the two major categories of information structure: declarative knowledge and procedural knowledge. The third subsection discusses all three aspects of help architecture: integration of help module, separation of help knowledge from user interface knowledge, and application dependence.

Declarative Domain Knowledge Help

The heart of a help system is the knowledge about the application for which help is being provided. Those help systems which explicitly store retrievable tokens of information are categorized as having *declarative* knowledge structures. Help generation mechanisms for systems in this category mostly use information access algorithms whose sophistication depends on the degree to which information accessibility takes into account various contextual factors (user preferences, history, program state, etc.). The number of contextual factors used may also indicate how sophisticated help explanations are, if these factors are used to vary explanations.

Existing on-line help systems, such as Unix™ man pages, Microsoft Word help [Microsoft89], Elm mail utility help [Taylor89], etc., use linear text as the information source for help. Pre-stored linear text is a convenient format for technical writers to create a help database for an application; however, the format lacks functional descriptions of the information which would enable incorporation of context into help presentations utilizing these functional descriptions. This serious drawback is driving help researchers towards functionally representing information and incorporating contextual factors to intelligently reuse the representations.

As early as 1982, Mark represented information about what interactive systems can do in a central knowledge base of his Consul help system [Mark82]. The attempt marked an early use of declarative knowledge to store system terminologies acquired from application programmers to aid in assisting users. Johnson [Johnson84] explicitly stored detailed specifications of tasks and objects (on which actions in task descriptions operated) in his task knowledge descriptions, which were then used to explain procedural tasks. Both Mark and Johnson embodied declarative

procedural descriptions in their help systems—the knowledge of application systems which was defined independently of the applications themselves.

In the early 1980's, help features were also being proposed from the natural language point of view as evidenced in [Cullingford82]. His representations inherited characteristics of those used in Conceptual Dependency representations [Schank81], which centered around actions, their subjects and objects, and causal links to other actions. By using context to determine which casual link was appropriate, a chain of actions which achieved a task could be derived. Wilensky also used conceptual dependency representations in his Unix™ Consultant help system [Wilensky84]. However, instead of storing casual links, he stored effects of actions in his representations; thus a chain of actions could be more dynamically derived using a planning agent. The idea of using a planning agent to derive a task sequence is valuable, though one has to keep in mind that AI planning techniques are very representation-dependent.

These two research efforts showed a very promising direction for using a knowledge base to deliver English language textual explanations. However, their representations were rather static, and detailed components were centrally locked within single representations; changing interface styles of the same application would necessitate recapturing a major portion of the help knowledge base—an undesirable necessity in a prototyping environment.

In the EuroHelp project, Breuker declaratively stored both tasks and actions of an application domain [Breuker86]. In addition, he extensively represented objects of the application, and effects of actions were stated in terms of changes to object attributes. Kemke also represented actions, which she referred to as concepts, and objects, in her SINIX (Unix like) Consultant help system [Kemke87]. In addition to pre-condition information, her action representation included information about the pragmatic, semantic, and syntactic aspects of an action. By using her planning component, her help system was active in suggesting to users how to more efficiently perform the same task they had just performed.

Although the representations used in these research efforts may not resemble each other much internally, the kinds of entities the researchers chose to represent are similar, indicating which components are important in storing procedural task information. Both Mark and Johnson represented tasks with actions to perform them with no separate detailed representations of actions within the tasks. Breuker represented both tasks and actions. Cullingford represented individual actions with an ability to contextually derive task descriptions through casual links. Wilensky and Kemke represented conditions and effects of actions, which made dynamic derivations of task descriptions possible. In general, these researchers recognized the importance of representing actions as part of

tasks with their conditions and effects; this information is then used to generate procedural explanations dynamically at runtime.

An observation to make about the previous research is that these systems (with exception of Cullingford's) assumed a command-oriented interface to their applications; therefore, their representations never lend themselves to descriptions of the actual details of how to perform an action beyond assuming that certain pieces of information must be typed in. Their representational approaches are obviously not robust enough for innovative interfaces of today, which tend to engage multiple types of input devices with a corresponding variety of feedback mechanisms.

For a system with declarative knowledge structure, it is the granularity at which this information is stored and structured which aids (or prevents) a help generator to reason about which help explanations to produce. Storing information as linear text does not allow and does not provide access points for the help system to situationally retrieve relevant information, thus limits help generation to a mere display of text. When text is structured such that different points of access are allowed, a more fine-tuned situational access of different tokens of information allows a very limited form of sensitivity in help. Most context-sensitive help referred to in commercial applications, for example Microsoft Word help [Microsoft89], has this kind of limited sensitivity using application modes to encode points of information access [Microsoft89]. Help text is organized in units based on subtopics and users can directly point at a subtopic in a menu to access the related help text. This is the case of Microsoft Word, which organizes help into topics, each of which leads to several related subtopics. Help text or help information can be more finely structured based on existing entities (or objects) in an application such that only related information concerning an entity is stored with the entity. This approach, though it requires much more work at the application design time, is far more flexible. It requires the help generator to understand the function of entities in the application environment, which in turn allows help generation to compose help messages based on the functionality of the entities. Both linear text and entity-based information structures require that most of required help information be declared at design time. The help generator accesses the information and presents it with more or less sensitivity depending on how much it understands each token of information and its role.

Procedural Domain Knowledge Help

Those help systems which embed information as operational agents are included in the category of help systems with procedurally encoded information. There are several ways this has been done: *rule-based*, *grammar-based*, and *state-based* approaches, variously documented in [Genesereth82, Goldstein82, Fenchel82, Rich82, Fischer85]. Work in this area has considerably advanced the understanding of on-line help and guidance. Its major disadvantage is its domain dependence which renders these

systems' knowledge representations inapplicable to a new application domain.

A couple of example systems which encoded the domain knowledge as rules are in [Genesereth82] and [Fischer85]. Genesereth provided tutoring services for arithmetic procedures based on MACSYMA's procedural knowledge encoded in lisp procedures. Interestingly, these procedures were individualized to represent relevant atomic actions whose pre- and post-conditions were present as function variables. The true/false conditions of these variables determined further procedures (actions) to be invoked; the determining process was encoded within each function. Fischer encoded goals and how to achieve the goals in OPS5 rules; the same goals might be encoded multiple times for each different contextual condition.

An example of a system which encodes help knowledge in a grammar-based representation was detailed in [Fenchel82]. Their SARA system used states of the system's language parsing component as the knowledge source of on-line assistance. SARA's parse tables containing the interface grammar and a help database which linked syntactic and semantic help information with the parser's code were used by SARA's help system to generate assistance based on the parser states in processing user inputs. Their approach could support some domain independence; if new grammar rules were defined and the help knowledge base described, their help system would still function for the new grammar. The knowledge of the structure of the grammar rules was procedural, and was embedded in the parser, not in a sharable knowledge base.

The state-based help system developed by Goldstein [Goldstein82] did not represent help information in state diagram specifications per se. He implicitly encoded help generation mechanisms within program components—the mechanisms which were activated based mainly on the state of the program execution. Goldstein encoded paths of knowledge to be learned in a graph, where each node contained a rule to suggest what to be learned, and node links stated pedagogical relationships among these nodes. Depending on the development of individual student's knowledge relative to this graph, a tutoring path was derived by the system's pedagogical strategies. The approach is similar to knowing which program state the user is in and suggesting solutions from that state to a goal state.

Rich suggested using the program code itself as information source for help [Rich82]. Even though the program code she used was in the rule-based form, her approach is still categorized in our taxonomy as a state-based help system since knowing the state of the rule firings and providing assistance based on the forward chaining anticipation is somewhat similar to knowing which state a program is in and knowing the possible directions which the program can take. Obviously, her rule-based system is more complicated than other state-based systems. Rich's model, which has the advantage that rules in the program code are

the direct source of help and changes in the rules are reflected immediately in help, would be inappropriate for most of existing programming languages. Rules may contain partial semantic information useful for help; languages like C or Fortran use syntactic constructs and extracting semantic information from program code would be impossible. Also regardless of the programming language used, in an environment where interface code is separated or hidden from the actual application code, her approach provides no guidance for retrieving the right combination of the necessary help information.

For all help systems based on a procedural knowledge structure, information is embedded within the help generator and is reproduced every time help is generated. In the rule-based system, production rules, which fire based on the current state of user's understanding, store within the rules other information which the users need to know next in order to complete their task at hand. Grammar-based systems determine acceptable grammars within their parsers and state-based systems determine appropriate input states within their state diagram representations. These systems use the algorithmic information embedded in them to deduce relevant information specific to particular states to drive the help explanation generation procedure. The sophistication of a help system based on a procedural knowledge structure is measured by how parts of the help system program are invoked, possibly based on similar contextual factors used by the help systems in the declarative knowledge category, and by how help explanations are produced (not retrieved) from within these program parts.

With a couple of exceptions [Fenichel82, Rich82] among the systems mentioned above, procedural help information connotes domain dependence. A major drawback of this kind of help knowledge approach is its non-reusability; applying it to a different application domain normally means recoding procedures. This contradicts if our objectives are to incorporate help development into a rapid prototyping environment.

Help Architecture

Three distinguishing features among help architectures: *integration* with underlying user interface tools, *sharing knowledge* with the user interface tools, and *application dependence*, are indicated in Table 3. These features are used to compare help systems in terms of how much runtime information they can use to vary explanations, the accuracy and consistency of help knowledge relative to the applications they support, and their portability.

Building attentive, sensitive help must be well-planned so that its architecture is tightly integrated with the application it supports [Senay89]. Building help as an isolated subsystem is an ad hoc solution, and does not give the help system access to runtime information which enables help sensitivity. It thus fails as a long term solution to providing effective help. Help architecture, if tightly integrated with user interface tools, can share runtime information and current context information, and thus allow

for such information to be incorporated in generating context-sensitive help explanations. The SARA system [Fenichel82] could context-sensitively provide help for users about its textual interface mainly because its help system was integrated with the system's input parser. SARA's help system used the state of parsing to explain user mistakes and what would be correct.

If the information used by help systems can be dynamically and totally derived from that which is used to control the application interface, help is guaranteed to always provide accurate explanations. This kind of help architecture is said to share knowledge which drives user interface tools. The User Interface Design Environment (UIDE) [Foley91] and Cartoonist [Sukaviriya91] are examples of such a system. It compares the pre-conditions of an action with the current context and is able to explain why the action is disabled; the same action and pre-condition knowledge is used to control when to and when not to allow the user to perform certain actions. Cartoonist used pre- and post-conditions to vary animated demonstrations appropriate to the current context. In contrast are those help architectures which have help running as a separate program entity [Cullingford82, Senay87, Kemke87]. In such systems, maintenance work is necessary to ensure that application interface behavior is consistent with what help "knows" about the application interface.

Prototyping a sophisticated help system does not guarantee wide acceptance if reproducing such a system requires tremendous amount of extra programming work, in addition to that already required in the process of building the application itself. One approach to building some independence while reducing programming loads is to model a help support tool highly-integrated with a domain-independent user interface environment, such as UIDE [Foley91], which operates at a high-level semantic structure. Modeling help generation to operate at the same representational structure, while the representation contents can be filled specifically for each particular application, makes possible transfer of the generation mechanism from one application to another, hence preserves the application independence. At the same time, help can be generated specifically to each application in the same fashion a user interface can be tailored specifically to application semantics in a domain-independent user interface environment. This is the approach used in Cartoonist—to embed an application-independent context-sensitive animated help tool, as part of automatic runtime support architecture, which consistently delivers accurate procedural help.

CONCLUSIONS

We have developed a comprehensive taxonomy incorporating distinguishing characteristics from user interface technology, both from the user and the system points of view. The taxonomy is useful in at least two ways: 1) it systematically identifies independent axes along which help can be categorized, which in turn encloses a space of help categories in which to place currently existing

help research; 2) it identifies distinct help software architectural features which contrast pros and cons in implementing help systems. Hopefully, with this taxonomy, new help research can identify with previous research, and be able to build on top of it. Our goal is to alleviate the meta problem of lacking an appropriate taxonomy to define help information content, to model access to help methods, and to categorize strategies used to generate help – the problem which has prevented appropriate mappings from help problems to technically sound solutions.

ACKNOWLEDGEMENT

We also would like to thank the graphics and user interface research group at the George Washington University, where most of this research was done, for the intellectual environment they provided. This research is funded by the National Science Foundation Grant IRI-88-13179, Sun Microsystems, Inc., Siemens Corporation, and the Software Productivity Consortium.

REFERENCES

- [Apple88] Apple Computer, Inc. *Macintosh® System Software User's Guide Version 6.0*. Cupertino, California: Apple Computer, Inc, 1988.
- [Balzert87] Balzert, H., and R. Lutze. "Information and Consultation Systems—A New Dimension of User Support." In *Proceedings INTERACT'87, 2nd IFIP Conference on Human-Computer Interaction*. 1987, 173-178.
- [Barnard87] Barnard, P., M. Wilson, and A. Maclean. "Approximate Modelling of Cognitive Activity: Towards an Expert System Design Aid." In *Proceedings of the Human Factors in Computing Systems and Graphics Interface*. April 1987, 21-32.
- [Bieger87] Bieger, G.R. and M.D. Glock. "The Information Content of Picture-Text Instructions," *Journal of Experimental Education*, 53(2), pp. 68-76, 1985.
- [Borenstein85] Borenstein, N.S. *The Design and Evaluation of On-Line Help Systems*. A Ph.D. Dissertation, Carnegie-Mellon University, 1985.
- [Breuker86] Breuker, J., R. Winkels, and J. Sandberg. "Didactic Goal Generator." *Deliverable 2.2.3, ESPRIT Project P280 'EUROHELP*. University of Amsterdam, December 1986.
- [Breuker87] Breuker, J. "Coaching in Help Systems." In *Intelligent Computer-Aided Instruction*, ed. J. Self. London: Chapman & Hall, 1987.
- [Carroll87] Carroll, J.M., and J. McKendree. "Interface Design Issues for Advice Giving Expert Systems." *Communications ACM* 30 (January 1987): 14-31.
- [Carroll88] Carroll, J.M., and A.P. Aaronson. "Learning by Doing with Simulated Intelligent Help." *Communications ACM* 31 (September 1988): 1064-1079.
- [Chin86] Chin, D.N. "User Modeling in UC, the UNIX Consultant." In *Proceedings of Proceedings of Human Factors in Computing Systems, CHI'86*. 1986, 24-28.
- [Cullingford82] Cullingford, R.E., M.W. Krueger, M. Selfridge, and M.A. Bienkowski. "Automated Explanations as a Component of a Computer-Aided Design System." *IEEE Transactions on Systems, Man and Cybernetics* 12 (March/April 1982): 168-181.
- [Feiner85] Feiner, Steve. "APEX: An Experiment in the Automated Creation of Pictorial Explanations." *IEEE Transactions on Computer Graphics and Applications* 5 (November 1985): 29-37.
- [Fenchel82] Fenchel, R.S., and G. Estrin. "Self-describing Systems Using Integral Help." *IEEE Transactions on Systems, Man and Cybernetics* 12 (March/April 1982): 162-167.
- [Fischer85] Fischer, G., A. Lemke, and T. Schwab. "Knowledge-based Help System." In *Proceedings of Human Factors in Computing Systems, CHI'85*. 1985, 161-167.
- [Foley91] Foley, J.D., W.C. Kim, S. Kovacevic, and K. Murray. "UIDE—An Intelligent User Interface Design Environment." In *Architectures for Intelligent Interfaces: Elements and Prototypes*. Eds. J. Sullivan and S. Tyler, Reading, MA: Addison-Wesley, 1991.
- [Genesereth82] Genesereth, M.R. "The Role of Plans in Intelligent Teaching Systems." In *Intelligent Tutoring Systems*. Eds. D. Sleeman and J.S. Brown, London: Academic Press, 1982.
- [Goldstein82] Goldstein, I.P. "The Genetic Graph: A Representation for the Evolution of Procedural Knowledge." In *Intelligent Tutoring Systems*. Eds. D. Sleeman and J.S. Brown, London: Academic Press, 1982.
- [Gong90] Gong, R., and J. Elkerton. "Designing Minimal Documentation Using a GOMS Model: A Usability Evaluation of an Engineering Approach." In *Proceedings of Human Factors in Computing Systems, CHI'90*. April 1990, 99-106.
- [Gwei90] Gwei, G.M., and E. Foxley. "Towards a Consultative On-line Help System." *International Journal of Man-Machine Studies* 32 (April 1990): 363-383.

- [Hecking87] Hecking, M. "How to Use Plan Recognition to Improve the Abilities of the Intelligent Help System SINIX Consultant." In *Proceedings INTERACT'87, 2nd IFIP Conference on Human-Computer Interaction*. 1987, 657-662.
- [Johnson84] Johnson, P., D. Diaper, and J. Long. "Tasks, Skills and Knowledge: Task Analysis for Knowledge Based Descriptions." In *Proceedings INTERACT'84, 1st IFIP Conference on Human-Computer Interaction*. 1984, 23-27.
- [Kearsley88] Kearsley, G. *Online Help Systems: Design and Implementation*. New Jersey: Ablex Publishing Corporation, 1988.
- [Kemke87] Kemke, C. "Representation of Domain Knowledge in an Intelligent Help System." In *Proceedings INTERACT'87, 2nd IFIP Conference on Human-Computer Interaction*. 1987, 215-220.
- [Lutze87] Lutze, R. "Customizing Help Systems to Task Structures and User Needs." In *Proceedings INTERACT'87, 2nd IFIP Conference on Human-Computer Interaction*. 1987, 871-878.
- [Mark82] Mark, W. "Natural-language Help in the Consul System." In *AFIPS Proceedings of the National Computer Conference*. 1982, 475-479.
- [McKendree88] McKendree, J., and J. Zaback. "Planning for Advising." *Proceedings of Human Factors in Computing Systems, CHI'88*. May 1988, 179-184.
- [Microsoft89] Microsoft Corporation. *Reference to Microsoft® Word: Microsoft Word Document Processing Program Version 4.0 for the Apple® Macintosh™*. Washington: Microsoft Corporation, 1989.
- [Neiman82] Neiman, D. "Graphical Animation from Knowledge." In *Proceedings of AAAI'82*. 1982, 373-376.
- [OPENLOOK90] Sun Microsystems, Inc. *OPEN LOOK™ Graphical User Interface Application Style Guidelines*. Reading, MA: Addison-Wesley Publishing Company, Inc., 1990.
- [Rich82] Rich, E. "Programs as Data for their Help Systems." In *AFIPS Proceedings of the National Computer Conference*. 1982, 481-485.
- [Robinson87] Robinson, C.P. "Comparison of Speech and Pictorial Displays in a Cockpit Environment." *Human Factors* 29,1 (1987): 31-44.
- [Schank81] Schank, R.C., and C.K. Riesback. *Inside Computer Understanding*. New Jersey: Lawrence Erlbaum Associates, 1981.
- [Senay87] Senay, H. *A Knowledge-based Approach to Designing Interfaces*. A Ph.D. Dissertation. Syracuse University, 1987.
- [Senay89] Senay, H., P. Sukaviriya, and L. Moran. "Planning for Automatic Help Generation." In *Proceedings of Working Conference on Engineering for Human Computer Interactions*. August 1989.
- [Stevens87] Stevens, C., and J. Miller. "Advice Objects: Direct Manipulation Techniques for Advisory Communication." *MCC Technical Report ACA-HI-294-87*. Microelectronics and Computer, Texas, September 1987.
- [Sukaviriya88] Sukaviriya, P. "Dynamic Construction of Animated Help from Application Context." In *Proceedings of the ACM SIGGRAPH User Interface Software Symposium*. October 1988, 190-203.
- [Sukaviriya91] Sukaviriya, P. *Automatic Generation of Context-Sensitive Animated Help* A Ph.D. Dissertation. Washington D.C.: The George Washington University, 1991.
- [Taylor89] Taylor, D. *The Elm Reference Guide*. Hewlett-Packard Laboratories, California, 1989.
- [Travellyan87] Travellyan, R., and D.P. Browne. "A Self-regulating Adaptive System." In *Proceedings of the Human Factors in Computing Systems and Graphics Interface*. April 1987, 103-107.
- [Wilensky84] Wilensky, R., Y. Arens, and D. Chin. "Talking to UNIX in English: An Overview of UC." *Communications ACM* 27 (June 1984): 574-593.