TIPS: A FRAMEWORK FOR CREATING SPECIALIZED TOOLS

Lloyd H. Nakatani and Laurence W. Ruedisueli

__Lloyd H. Nakatani__, a distinguished member of technical staff, and __Laurence W. Ruedisueli__, a member of technical staff, are in the Computer-Aided Information Systems Research Department at AT&T Bell Laboratories in Murray Hill, New Jersey. Mr. Nakatani's research encompasses computer-aided learning environments, human-computer interface design, and programming languages. He joined the company in 1968 and has both a B.A. in mathematics and a Ph.D. in psychology from the University of California at Los Angeles. Mr. Ruedisueli's interests include programming environments, and hardware and software to support human-computer interfaces. He joined the company in 1979, attended Rensselaer Polytechnic Institute, and is currently in a graduate program in computer science at Columbia University.

A complex, computer-based task that is difficult to do with a general-purpose tool can be facilitated by a specialized tool with capabilities tailored for the task. To make this approach to task facilitation attractive, we developed a software framework, called TIPS, that enables us to take an existing UNIX® system tool—a general-purpose text editor, for instance—and specialize it by augmenting its basic capabilities with new, task-specific capabilities. The original tool is not compromised by this specialization. As an application of the TIPS framework, we built a specialized text editor for text formatting on the UNIX system. The specialized tool provides a variety of on-line, context-sensitive services that make the task easier to learn, do, and remember. Because its services are always available and tailored to the situation at hand, the specialized tool compares favorably to conventional task supports, such as training classes and paper manuals, as a method for task facilitation.

Introduction

How can we help people cope with complex, computer-based tasks? One approach is to provide environments specialized for the tasks. The world around us seems to have evolved to this answer. Look at a house, for example. Each room is specialized for an activity: a kitchen for cooking, a bathroom for hygiene, an office or a shop for work, and so on. The office or shop may be further specialized depending on the nature of the work.

Specialized environments also exist on computers as *software tools*—for example, a text editor for working with text, a spreadsheet for working with tables of numbers, and so on. [In the personal computer (PC) world, software tools tend to be highly specialized and focussed on a single task—for example, a word processor for creating, formatting, and printing a document; a spreadsheet for computing data and printing reports; a database for organizing, storing, and retrieving information. Integrated packages are also available for doing several related tasks, particularly "office" tasks, so users have a single environment for doing a variety of tasks. This paper focusses on the UNIX system world, where you need a combination of tools to perform a task—for example, one tool to create or change the text, another to format it.]

If specialization is a good thing, why are most software tools general-purpose? Why, for instance, is there only a general-purpose text editor such as `vi` on most UNIX system machines instead of many specialized text editors: one for writing shell scripts, another for writing C programs, yet another for writing business letters, and so on?

An obvious reason is that tools cost money to buy or build. This is the *acquisition cost.* Another reason is that specialization leads to a proliferation of tools and thus increases the cost of learning, remembering, and mastering their use. We'll call this the *competency cost.* Another reason is *flexibility loss.* A tool specialized for one task is often awkward to use for other tasks. Once we start down the road of specialization, we may find that we need a different tool for each task, which brings us up against acquisition and competency costs.

We have developed a framework—called TIPS—for building specialized tools that lowers acquisition cost, minimizes competency cost, and eliminates flexibility loss entirely.

TIPS lowers the acquisition cost of a specialized tool by starting with an existing tool as a base and enhancing it with task-oriented, on-line services. The tool we would choose as the base tool is one that is already in common use for the task and that a user would learn as a matter of course. The base tool provides the core capabilities of the specialized tool, so only the enhancements need to be learned. This minimizes competency cost, and overcomes the reluctance of users to learn an entirely new tool from scratch. In addition, several specialized tools may use the same base tool, but with the same interface for the enhancements. TIPS eliminates flexibility loss because the enhancements don't compromise any of the existing capabilities of the base tool.

In the next section, we describe a specialized tool that was developed with TIPS for formatting text. This is followed by a brief description of the TIPS framework and a discussion of the user-interface benefits of this approach.

A Specialized Tool for Text Formatting

To explore the benefits of a specialized tool and the TIPS framework, we built a specialized tool for formatting text using MM, that is, the UNIX system's memorandum macros.[1] Before we describe the tool, let's first understand what the task involves and why it serves our purposes well.

MM is a *markup language* for specifying the format of a document. It consists of a code language that is added to text to tell how particular parts of the text should be handled. (See Panel 1 for an example of how MM is used to produce a letter.) Strictly speaking, `mm` is the name of a software tool that generates a well-formatted document, given the proper input. Preparing the input for the `mm` tool—that is, inserting the correct format information into the text—is a task, just as writing a program for a compiler is a task. From now on, when we refer to MM, we mean the task of preparing a document for input to the `mm` text-formatting software tool, unless the context suggests some other meaning.

The following aspects make MM a good task for exploring the usefulness of a specialized tool:

- MM is complex. Classroom instruction on MM takes six days, testimony to its complexity. Entire books have been written on MM and related topics.[1,2] Coping with the complexity of MM should be a challenge for a TIPS tool.
- MM is a real task in the real world. Thousands of people already use MM. Each year, several hundred students take the MM course, and many more learn on their own. Given this need, a useful TIPS tool should

Panel 1. MM: A Brief Overview

Formatting instructions or *macros* in MM consist of the macro's name and its argument or arguments. Consider, for example, the macro `.WA "J. Doe"` that provides information about the writer of a letter. (We'll explain this macro further shortly.) Its name is `.WA`, and its argument is `"J. Doe"`. (Quotation marks must surround any argument that contains embedded spaces.) Macros are distinguished from normal text by a period (pronounced *dot*) at the beginning of the line.

We now present a brief example to show how the memorandum macros (MM) are used to produce the following letter:

J. Doe
AT&T Bell Laboratories
Murray Hill, NJ 07974

Equipment Supplier, Inc.
1234 Main Street
Corpville, NJ 01234

To whom it may concern:

Please send me information about your Widget, Model A123. Thank you.

J. Doe

This letter results from formatting and printing the following MM document file:

```
.WA "J. Doe"
AT&T Bell Laboratories
Murray Hill, NJ  07974
.WE
.IA
Equipment Supplier, Inc.
1234 Main Street
Corpville, NY  01234
.IE
.LO SA
.LT BL

Please send me information
about your Widget, Model A123.
Thank you.
.SG
```

This is a brief description of the letter's components. The *writer's address* macro, `.WA`, specifies the writer's name—e.g., `J. Doe`—and denotes that the writer's address appears on the lines that follow. The `.WE` macro marks the end of the writer's address. The *recipient's address* macros, `.IA` and `.IE`, play similar roles for the recipient of the letter. The *letter option* macro, `.LO`, with the code, `SA`, for its argument generates the salutation, "`To whom it may concern:`" The *letter style* macro, `.LT`, with the code, `BL`, for its argument specifies a style with blocked paragraphs. The *signature* macro, `.SG`, generates the signature line at the bottom of the letter.

find many users. If not, the failure will be obvious.

- There is competition. Classes, manuals,[3,4] books,[1,2] and computer-aided instruction (the UNIX system's `teach` and `learn` facilities)[5] are currently available for learning MM. The extent to which a TIPS tool succeeds against this competition is a measure of its relative usefulness.

Our goal was to explore the TIPS framework, not to permit or simplify the formatting of text, per se. If the latter were our goal, we might redesign the text-formatting task to eliminate the root causes of the difficulties with MM.

A good example of a redesigned task is a WYSIWYG (what you see is what you get) word processor that displays the formatted document as you create it, unlike MM where the document is formatted only when it is printed. Does the possibility of redesigning tasks make specialized tools unnecessary, or, worse yet, misguided? We think not.

Task redesign won't eliminate the need for help. Many PC word-processing packages follow the WYSIWYG approach but mastering or even learning to use any of them is not easy. Witness the number of books available on how to use the so-called, easy-to-use WYSIWYG word processors on the market. Many such word processors also include on-line help. We take these observations as evidence that users need support in dealing with complex tasks, no matter how well designed the tasks.

Moreover, redesign may make a task more complex than before. The tendency is to make each generation of a tool more powerful, and the tasks done with the tool more complex. As long as this tendency continues, a role for task support seems assured.

Using the TIPS framework, we built a specialized tool for MM with a general-purpose text editor as its base tool. A text editor is an appropriate choice for the base tool, because it is normally used to create and edit MM documents.

The text editor can be either the `vi` or `emacs` screen editor. (`Vi` is pronounced *vee eye*, while `emacs` is pronounced *ee max*.) If you use the UNIX system, then you probably know one of them already because a text editor is such an essential and heavily used tool.

For the sake of this paper, we'll assume that you use `vi`, and we'll use `vi`$_{MM}$ to denote the version specialized for MM. Because `vi`$_{MM}$ retains all the text editing capabilities of `vi`, you already know how to use most of `vi`$_{MM}$. You can still use `vi`$_{MM}$ just like plain `vi`. But when you format a document with MM, you can take advantage of the task-oriented, on-line services that `vi`$_{MM}$ provides to make the task easier:
- On-line documentation
- Sample documents
- Help with MM macro syntax (called forms)
- Plans for various types of documents
- A notebook for personal samples of documents.

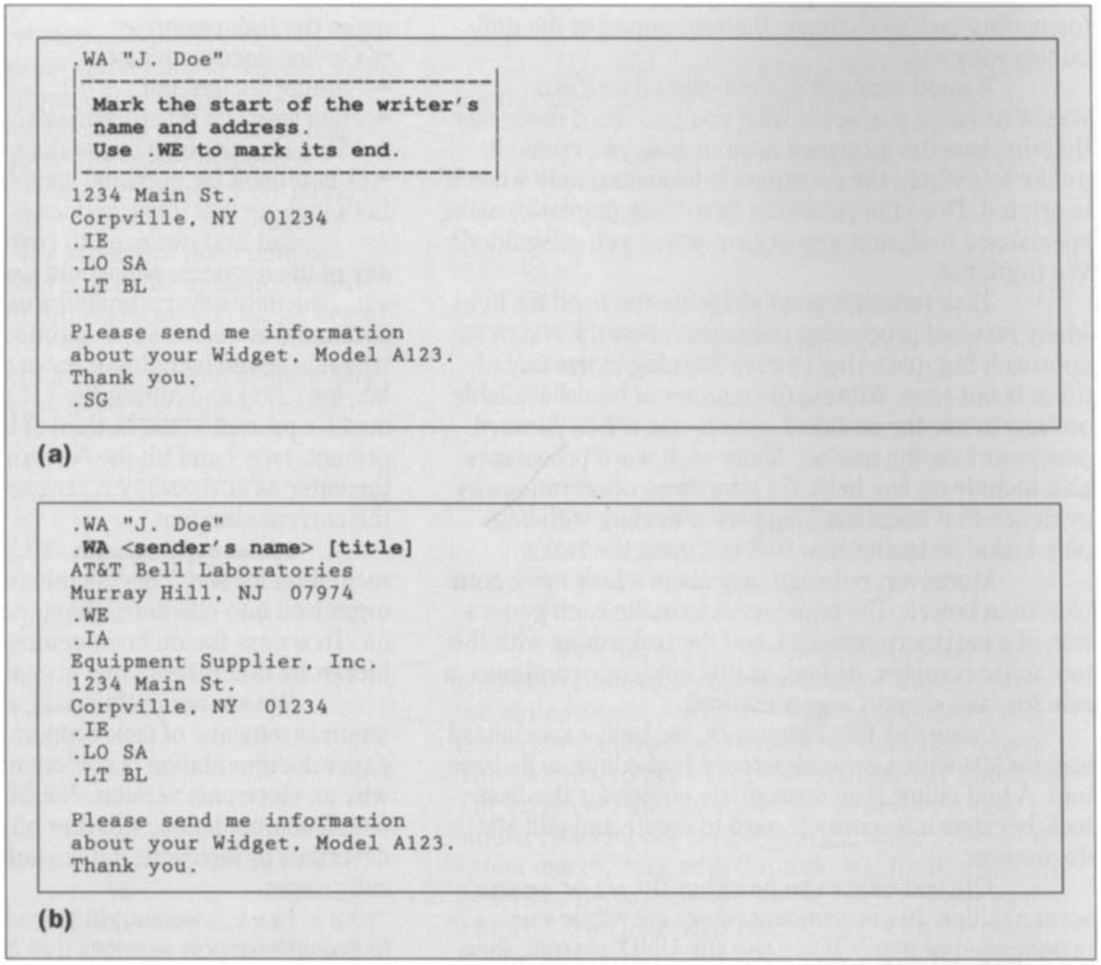
Let's look next at these services.

But first, we need to cover one detail. To get to any of the services, you would use the *help* command in `vi`$_{MM}$, the only new command a user need learn. The help command is `control-/` (pronounced *control slash*), which is keyed by holding down the "control" key (often labeled CTRL) and hitting the "/" key. This command produces a prompt at the bottom of the screen. At the prompt, type ? and hit the "return" key. (We'll refer to the latter as RETURN.) `Vi`$_{MM}$ responds with help tailored to the current situation.

On-Line Documentation. `Vi`$_{MM}$ offers on-line documentation for MM. This documentation is hierarchically organized into chapters, chapters into sections, and so on. To access the on-line documentation, you use a menu hierarchy that reflects this organization.

We are not enthusiastic about on-line documentation as a means of task support. Studies show that paper documentation is seldom used,[6-8] and we question why an electronic version should not suffer the same fate. Documentation, whether on paper or screen, only describes or instructs; it does not make the task intrinsically easier.

In `vi`$_{MM}$, we sought to go beyond documentation to provide support services that are possible only because a computer can sense and respond to the current situation. For example, suppose you want an explanation about some detail of MM that appears on the text editor screen. You can "point" to the puzzling item with the text editor's cursor and then use the help key to ask `vi`$_{MM}$ for an explanation. (To *point to* or *select* an item on the screen, move the cursor to that item with the text editor's normal cursor-control commands and then use the help key.) `Vi`$_{MM}$ will "sense" what you have pointed to,



```
.WA "J. Doe"
|---------------------------------|
| Mark the start of the writer's  |
| name and address.               |
| Use .WE to mark its end.        |
|---------------------------------|
1234 Main St.
Corpville, NY  01234
.IE
.LO SA
.LT BL

Please send me information
about your Widget, Model A123.
Thank you.
.SG
```

(a)

```
.WA "J. Doe"
.WA <sender's name> [title]
AT&T Bell Laboratories
Murray Hill, NJ  07974
.WE
.IA
Equipment Supplier, Inc.
1234 Main St.
Corpville, NY  01234
.IE
.LO SA
.LT BL

Please send me information
about your Widget, Model A123.
Thank you.
```

(b)

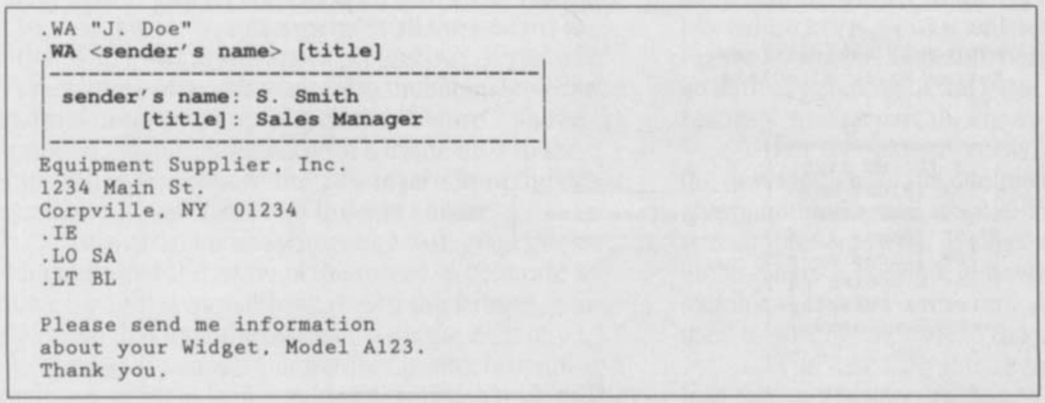
Figure 1. On-line documentation. (a) An explanation for the .WA macro; (b) the raw form of the macro; (c) the setter allows you to change the macro's arguments; (d) setter values replace the arguments in the macro's raw form. (For this publication, we have emboldened the critical information on "screens"; these areas are not emboldened on a real screen.)

search its stored knowledge about MM, and present an explanation on the screen. You have only to make your desire known, and the computer responds appropriately.

The remaining services described in this section show how `vi`$_{MM}$ provides assistance tailored to your needs and the situation at hand.

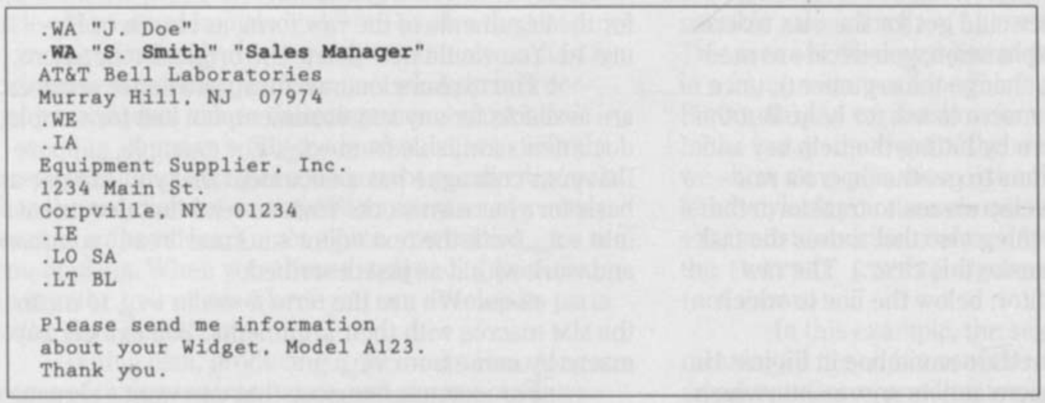
Sample Documents. A common strategy for solving a problem is to find an example of a solution to a similar problem, analyze the example to understand the solution, and then modify the example to solve your variant of the problem.[9] `Vi`$_{MM}$ lets you apply this strategy to learn and work with MM.

`Vi`$_{MM}$ contains a collection of sample documents that represent solutions to a variety of formatting problems. To get a sample document, use the help key. At the prompt, type ? and RETURN. A menu of all the services



```
.WA "J. Doe"
.WA <sender's name> [title]
|--------------------------------------|
 sender's name: S. Smith
       [title]: Sales Manager
|--------------------------------------|
Equipment Supplier, Inc.
1234 Main St.
Corpville, NY  01234
.IE
.LO SA
.LT BL

Please send me information
about your Widget, Model A123.
Thank you.
```

(c)



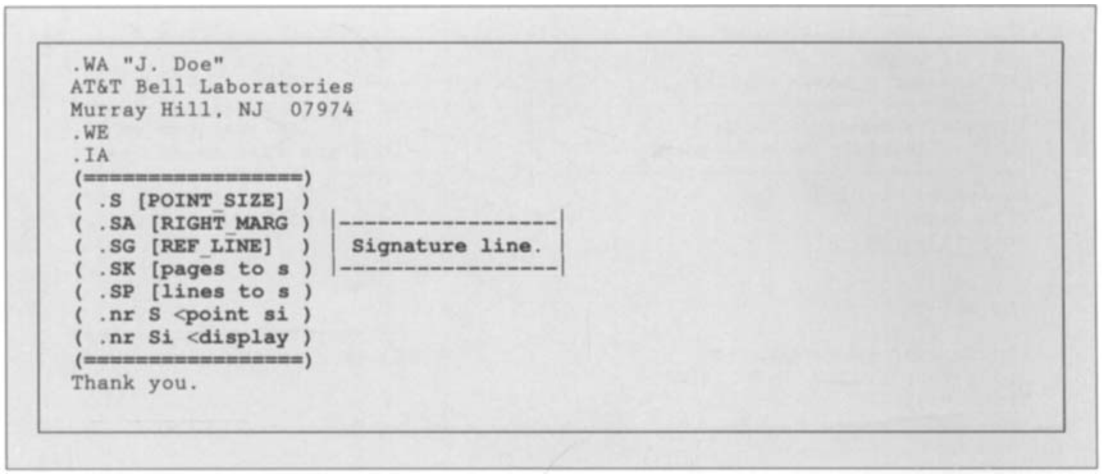
```
.WA "J. Doe"
.WA "S. Smith" "Sales Manager"
AT&T Bell Laboratories
Murray Hill, NJ  07974
.WE
.IA
Equipment Supplier, Inc.
1234 Main St.
Corpville, NY  01234
.IE
.LO SA
.LT BL

Please send me information
about your Widget, Model A123.
Thank you.
```

(d)

available will appear on the screen. From the menu, select `Examples`. In response, vi$_{MM}$ will present a menu of sample documents. Then, when you choose one, that sample document is read into the text editor.

Now you can analyze the sample with vi$_{MM}$'s help and use the text editor to change the sample to meet your needs. This service of vi$_{MM}$ enables you to learn from examples and do useful work as you learn.

For example, suppose you want to write a letter. You might start with the coded sample letter we saw in Panel 1, which is available from vi$_{MM}$. If there is something you don't understand about MM in the sample— for instance, the `.WA` macro—you can point to that line with the text-editor cursor and ask vi$_{MM}$ for an explanation. The explanation is presented on the screen in a "frame" placed directly below the line in question. Figure 1a



```
.WA "J. Doe"
AT&T Bell Laboratories
Murray Hill, NJ  07974
.WE
.IA
(================)
( .S [POINT_SIZE] )
( .SA [RIGHT_MARG )   |-----------------|
( .SG [REF_LINE]  )   | Signature line. |
( .SK [pages to s )   |-----------------|
( .SP [lines to s )
( .nr S <point si )
( .nr Si <display )
(================)
Thank you.
```

Figure 2. The desired macro can be found from just the first letter of its name. If you ask for a list of all the macros that begin with the letter `S`, you will get the menu on the left. A brief explanation of each item is available; the frame to the right displays an explanation of the `.SG` macro.

shows the explanation you would get for the `.WA` macro.

If, based on the explanation, you decide to modify the `.WA` macro (that is, change the argument), once again you can point to the macro to ask for help. But this time, *develop* the `.WA` macro by hitting the help key and RETURN. Here, develop means to get the macro's *raw form*. (More generally, develop means to transform the thing pointed to into something else that moves the task along; later examples will make this clear.) The raw form is put into the text editor, below the line to which you were pointing.

As we can see from the second line in Figure 1b, the raw form shows the macro and its arguments, which are enclosed in either angle ("<" and ">") or square ("[" and "]") brackets. Angle brackets mean the argument is required and a value must be supplied; square brackets signify it is an optional argument and may be ignored. (The brackets are omitted when the macro is typed into the text file.)

If you now point to the raw form and ask vi$_{\text{MM}}$ to develop it, you get the *setter*, a service that enables you to supply values for the arguments. In Figure 1c, we show the setter with new values already typed. When you finish with the setter, the values you typed are substituted for the arguments of the raw form, as shown in Figure 1d. You would now delete the original `.WA` macro.

The explanation, raw form, and setter services are available for any MM document, not just for sample documents available from vi$_{\text{MM}}$. For example, suppose that your colleague has a document that you can use as a basis for your own work. You can read that document into vi$_{\text{MM}}$ (with the text editor's normal "read" command) and work with it as just described.

Forms. We use the term *forms* in vi$_{\text{MM}}$ to mean the MM macros with their arguments. You can get any macro by name from vi$_{\text{MM}}$.

For example, suppose that you want a signature on a letter, but the letter you are creating does not have the `.SG` macro that generates the signature line. If you know that you need the `.SG` macro but aren't sure about its arguments, you can ask vi$_{\text{MM}}$ for the macro by name. (Use the help key and type the macro name, `SG`, instead of ? at the prompt.) Vi$_{\text{MM}}$ responds by putting a raw form of the macro into the text editor. With this service, you need to know only the name of a macro and not its syntax, because vi$_{\text{MM}}$ will supply the complete macro.

Here's a more interesting variation of the forms service. Suppose you want the signature macro but remember only that its name begins with an `S`. You can ask $\texttt{vi}_{\text{MM}}$ simply for `S`, and a menu of all the macros in MM that begin with `S` will appear (Figure 2). If you still can't recognize what you want from the menu, you can get a brief description of each choice. Figure 2 shows the description for the `.SG` macro (in a frame next to the menu). When you choose the `.SG` macro from the menu, $\texttt{vi}_{\text{MM}}$ will put its raw form into the text editor.

This variation lets you get by with good guesses. If you guess that the name of the macro to generate a signature might reasonably start with the letter `S`, you can explore that hypothesis easily with the help of $\texttt{vi}_{\text{MM}}$.

Plans. If you are a newcomer to MM, formatting a complete document is like putting together a puzzle. The individual MM macros are the pieces (declarative knowledge), and the process of formatting a document is the puzzle (procedural knowledge).[9,10] To solve the MM puzzle, you need not only the MM macros, but also a *plan* for the document: which macros to use, what their order must be, and what values their arguments must have.

$\texttt{Vi}_{\text{MM}}$ contains ready-made plans to help with the process of putting together a document puzzle. A *plan* is either the structure of an entire document or of a component of a document. To get a plan, you ask $\texttt{vi}_{\text{MM}}$ for a menu of plans. When you choose a plan, $\texttt{vi}_{\text{MM}}$ puts that plan into the text editor where you can develop its parts into the final document.

With a plan, producing a document is no longer a process of typing macros names and argument values from memory onto a blank screen. Instead, it is one of altering a ready-made plan bit by bit with $\texttt{vi}_{\text{MM}}$'s help, until the final document evolves. This is an application of the *means-ends* analysis,[9] where $\texttt{vi}_{\text{MM}}$'s ability to develop a plan is the principal means, and the final document is the goal.

For example, suppose that you want to produce a standard business letter. Figure 3a show the plan you get from $\texttt{vi}_{\text{MM}}$ for the letter with the expected three parts: the beginning, the body, and the end.

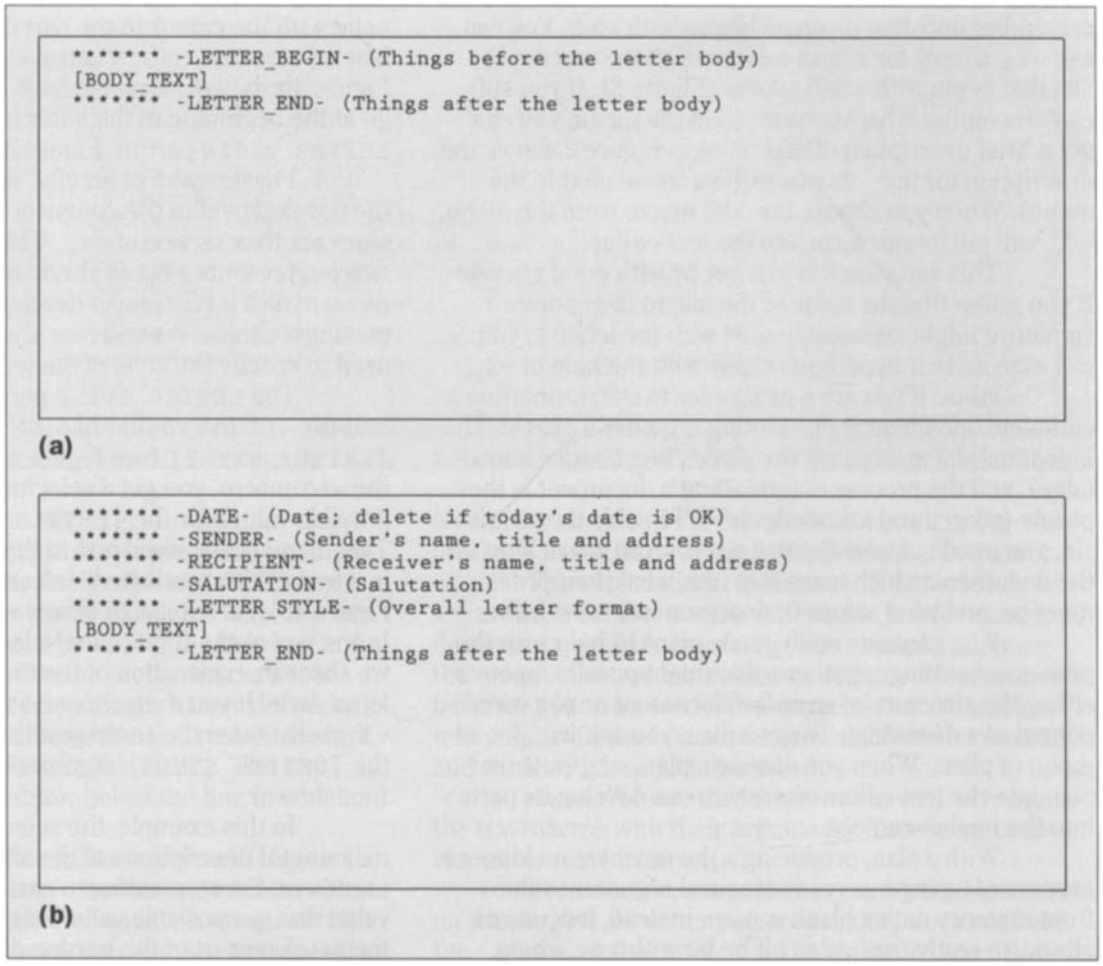
To format the beginning of the letter, you can point with the cursor to the part of the plan that is labeled `LETTER_BEGIN` and ask $\texttt{vi}_{\text{MM}}$ to develop it. Figure 3b shows the result; here, all the subparts that go at the beginning of the letter have replaced the `LETTER_BEGIN` part (in Figure 3a).

For the sake of brevity, we'll skip some steps in the development of the complete plan for this letter to show another service of $\texttt{vi}_{\text{MM}}$. This service, called the *selector*, presents a list of alternatives as a menu. The menu makes it possible to develop something by simply making a choice. We will now show how the selector is used to specify the style of the letter.

The `LETTER_STYLE` part of the plan develops into the `.LT` macro that has the argument named `[LETTER_STYLE]` (see Figure 4a). When you develop the `.LT` macro, you get a selector menu that lists the possible values for the `[LETTER_STYLE]` argument. This menu is displayed just to the right of the argument in a frame with "scalloped" left and right margins (see Figure 4b). An explanation for each value is available. In the box to the right of the selector menu in Figure 4b, we show an explanation of the first choice: the `Blocked` letter style. If you then choose `Blocked` as the style, $\texttt{vi}_{\text{MM}}$ substitutes the corresponding code value `BL` for the `[LETTER_STYLE]` argument in the `.LT` macro (not shown).

In this example, the selector menu consists of meaningful descriptions of the alternative values for an argument. Corresponding to each description is a code value that is a possible value for the argument. A selector menu relieves us of the burden of remembering lists of meaningless code values.

Notebook. $\texttt{Vi}_{\text{MM}}$ provides a private notebook as a convenient place to keep items for your personal use. such as a letter template with you already identified as the sender. If you develop such a template for a letter and want to save it for future use, you can store it in your private notebook with a name that you specify. Then, the next time you want to write a letter, you can ask $\texttt{vi}_{\text{MM}}$ for a menu that lists the names of items in your notebook,



```
******* -LETTER_BEGIN- (Things before the letter body)
[BODY_TEXT]
******* -LETTER_END- (Things after the letter body)
```

(a)

```
******* -DATE- (Date: delete if today's date is OK)
******* -SENDER- (Sender's name, title and address)
******* -RECIPIENT- (Receiver's name, title and address)
******* -SALUTATION- (Salutation)
******* -LETTER_STYLE- (Overall letter format)
[BODY_TEXT]
******* -LETTER_END- (Things after the letter body)
```

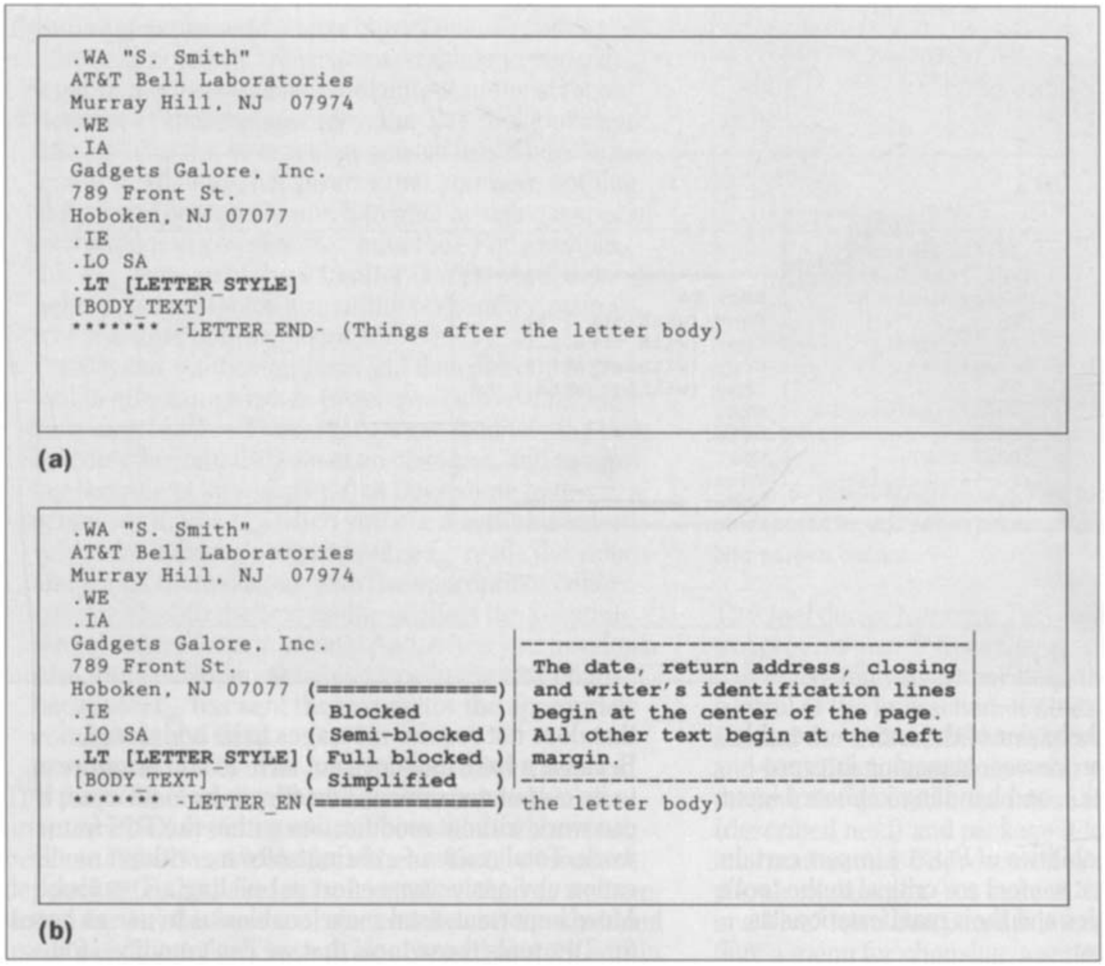
(b)

Figure 3. Ready-made plans represent the structure of the entire document or of one of its components.
(a) The plan for a business letter;
(b) the development of the beginning of a business-letter plan.

choose your letter plan to get it into the text editor, and get a head start on the letter.

The notebook is a convenience that relieves your memory, and is also a way to customize `vi`$_{MM}$ to suit your needs. Figure 5a is an example of a personalized template for a letter of the sort you might keep in your notebook. Notice that the template is not fully fleshed out. Instead, some macros are still in their raw forms, to be developed later when you use the template to write an actual letter.

For example, suppose you sometimes want to include a notation such as "Copy to" in your letter. You can develop the notation macro, `.NS`, to get the selector menu of possible notations in Figure 5b. Each notation is associated with an arbitrary code number that will replace the argument `[NOTATION_TYPE]` when you



(a)

```
.WA "S. Smith"
AT&T Bell Laboratories
Murray Hill, NJ  07974
.WE
.IA
Gadgets Galore, Inc.
789 Front St.
Hoboken, NJ 07077
.IE
.LO SA
.LT [LETTER_STYLE]
[BODY_TEXT]
******* -LETTER_END- (Things after the letter body)
```

(b)

```
.WA "S. Smith"
AT&T Bell Laboratories
Murray Hill, NJ  07974
.WE
.IA
Gadgets Galore, Inc.                  |-----------------------------------|
789 Front St.                         | The date, return address, closing
Hoboken, NJ 07077 (==============)   | and writer's identification lines
.IE               ( Blocked       )   | begin at the center of the page.
.LO SA            ( Semi-blocked )   | All other text begin at the left
.LT [LETTER_STYLE]( Full blocked )   | margin.
[BODY_TEXT]       ( Simplified    )   |-----------------------------------|
******* -LETTER_EN(==============) the letter body)
```

Figure 4. Specifying the document's style. (a) The business-letter plan developed through the `.LT` macro that controls the document's style. (b) The selector presents a menu of possible values for `[LETTER_STYLE]`. To the right is the explanation for the `Blocked` style.

make a choice from the menu. By keeping macros in their raw form, you leave open opportunities for vi$_{MM}$ to continue to help you.

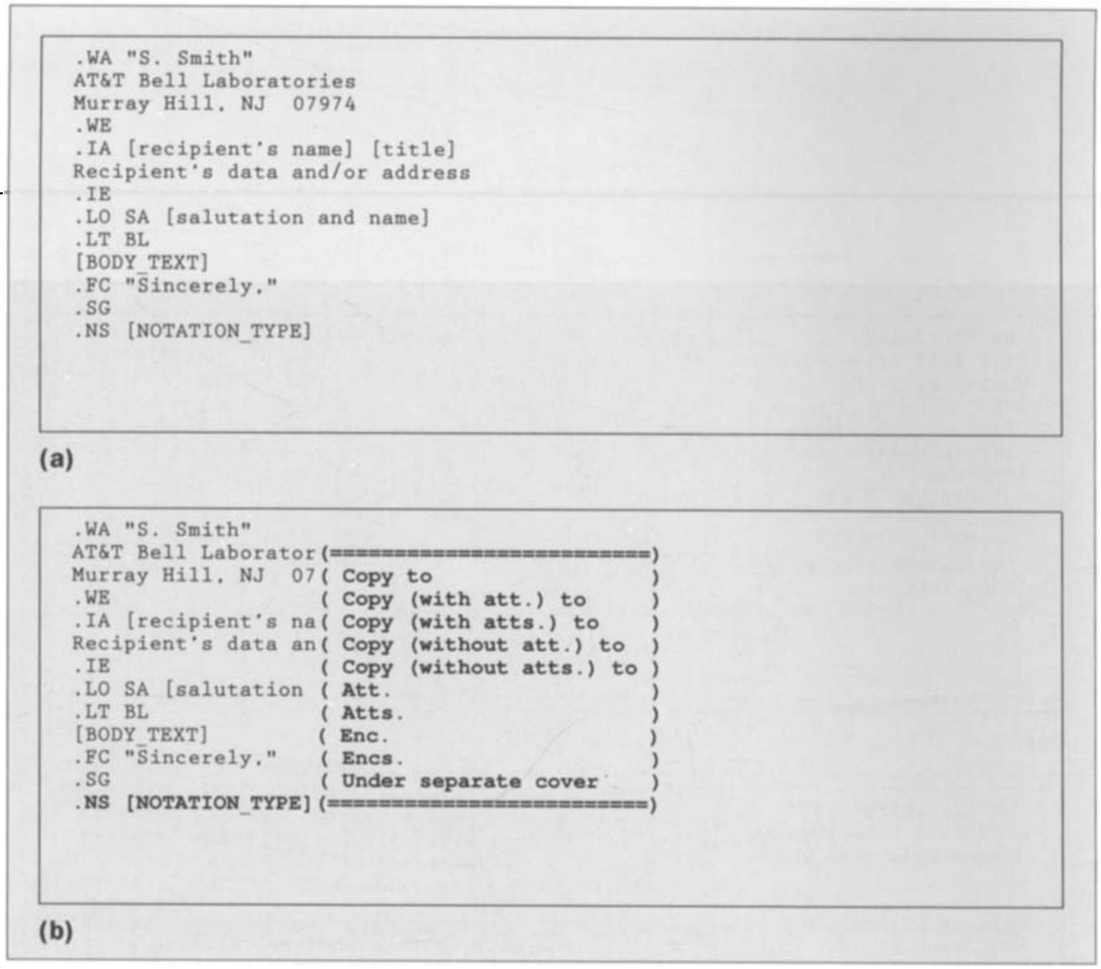
The TIPS Framework

A specialized tool built to comply with the TIPS framework consists of four major parts (Figure 6):

- The PLEXUS process.
- The base-tool process. (The base tool in our example above was the vi text editor.)
- The TIPS control process, which enhances the base tool with the task-specific services.
- The task database.

A short description of each part follows.

PLEXUS. The heart of the TIPS framework is the PLEXUS process, which plays an administrative role.



(a)

```
.WA "S. Smith"
AT&T Bell Laboratories
Murray Hill, NJ  07974
.WE
.IA [recipient's name] [title]
Recipient's data and/or address
.IE
.LO SA [salutation and name]
.LT BL
[BODY_TEXT]
.FC "Sincerely,"
.SG
.NS [NOTATION_TYPE]
```

(b)

```
.WA "S. Smith"
AT&T Bell Laborator(========================)
Murray Hill, NJ  07( Copy to                )
.WE                ( Copy (with att.) to    )
.IA [recipient's na( Copy (with atts.) to   )
Recipient's data an( Copy (without att.) to )
.IE                ( Copy (without atts.) to )
.LO SA [salutation ( Att.                   )
.LT BL             ( Atts.                  )
[BODY_TEXT]        ( Enc.                   )
.FC "Sincerely,"   ( Encs.                  )
.SG                ( Under separate cover   )
.NS [NOTATION_TYPE] (========================)
```

Figure 5. A notebook feature stores personalized "sample" documents. (a) A skeletal, personal version of a business letter may still contain raw forms of macros for later development. (b) This selector menu lists possible values for the `[NOTATION_STYLE]` argument.

PLEXUS coordinates the behavior of the entire ensemble by managing the other processes, managing interprocess communications (IPC), and handling keyboard input and screen output.

Some of the capabilities of PLEXUS impart certain features to a TIPS tool that we feel are critical to the tool's success. These capabilities and their manifestations as features of a TIPS tool are:

- To the base-tool process, PLEXUS looks like a terminal. Thus, PLEXUS can interpose itself between the terminal and a base-tool process that normally expects to be connected directly to a terminal. This ability of PLEXUS to mimic a terminal makes it possible to use screen-oriented interactive tools, such as the `vi` text editor, as the base tool.
- PLEXUS enables processes to communicate with each other via an interprocess communication mechanism that does not require messages to be addressed. Because a base tool does not have to add an address to its output nor remove an address from its input, it can work without modification within the TIPS framework. Total reuse of existing software without modification obviously saves effort in building a TIPS tool.[11] More important, total reuse enables us to use as bases for TIPS tools those tools that we can't modify—for example, tools from third-party vendors.
- PLEXUS maintains an exact copy of the screen. Because the information on the screen defines the context of the work in progress, a TIPS tool can use that information to respond intelligently to requests for help. That is, a TIPS tool can provide *context-sensitive* help. For example, when you point to a macro in $\texttt{vi}_{\text{MM}}$ and ask for an explanation, $\texttt{vi}_{\text{MM}}$ "reads" the text from the screen copy to determine what macro

you want explained.

- PLEXUS is normally transparent, sending keyboard input to the base tool and tool output to the screen. Because of this transparency, the TIPS tool looks and acts just like the base tool in normal use. There is no loss of flexibility. This assures that you have nothing to lose and potentially much to gain by using a specialized variant of your familiar base tool. For example, the `vi`$_{MM}$ version of the `vi` editor is a TIPS tool that helps you use MM for formatting text, unlike plain `vi`, which knows nothing about MM.
- PLEXUS can send commands and data directly to the tool to effect the work in progress. This is what Judd calls *direct help*.[12] Thus, PLEXUS can facilitate the task by going beyond the role of an observer, and assuming the role of an assistant that does some of the actual work. In `vi`$_{MM}$, when you use a setter to enter values for a macro's arguments, `vi`$_{MM}$ reads the values and sends them—along with the appropriate editor commands—to the text editor to effect the substitution of values for arguments. And, when you develop a plan, the development takes place in the text editor because `vi`$_{MM}$ has sent the text editor the appropriate commands and data.

The Base Tool. The base tool provides most of a TIPS tool's functionality. Both file-oriented and screen-oriented UNIX system tools can be used as a base tool, because PLEXUS can look like either a file or a terminal, depending on the base tool's orientation. And as we mentioned earlier, the base tool can be taken off the shelf and used "as is"; it doesn't have to be changed to work within the TIPS framework.

For now, the base tool must be character-oriented to work with TIPS. On one hand, this is an advantage because you don't need a fancy graphics terminal to use a TIPS tool. But on the other hand, this will prove increasingly to be a disadvantage as graphics terminals become more common and more UNIX system tools become graphics-oriented.

The Control Process. All task-specific services of a

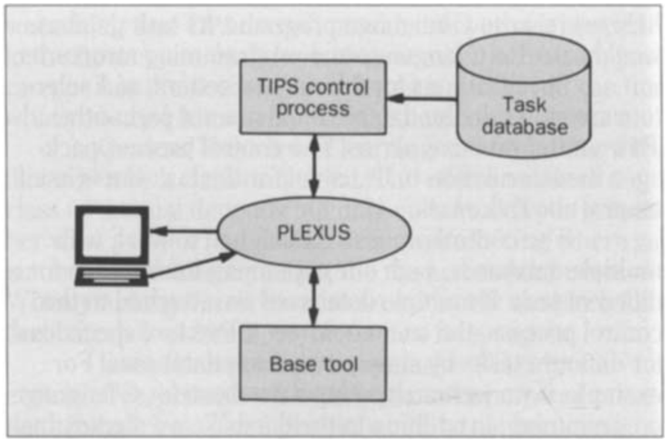


Figure 6. Block diagram of a TIPS tool. PLEXUS manages processes, interprocess communication, and keyboard input and screen output.

TIPS tool derive from the TIPS control process. The control process also defines the interface to the TIPS tool.

When you ask for help, the control process takes control of the interaction. It notes the position of the cursor and the text on the corresponding line on the screen, and uses this information—with your request—to retrieve the desired information from the database (described next) and package it for presentation. The information may be presented as a text box—for example, an explanation—to be read. Or, it may be presented in a "live" form that permits interaction with the information: a menu for choosing, a setter for typing, or a plan for developing. Or, the information may bypass the interface and go instead to the base tool to give direct help.

The Task Database. The task database attached to the control process contains all task-specific information, including information that specifies the interface for the specialized tool. In `vi`$_{MM}$ (and in the MM version of `emacs`), for example, this database contains information about the MM macros and how that information is to be presented. If a specialized TIPS tool were developed to help users write C-language programs, its task database would relate to C language and programming structures.

Specifications for the menus, setters, and selectors are stored in the database and are not part of the TIPS control process, per se. The control process packages the information to be useful for the task but is itself neutral about the task.

The control process is designed to work with multiple databases, each one containing information for a different task. If multiple databases are attached to the control process, the user could get a TIPS tool specialized for different tasks by simply switching databases. For example, if we were to develop a database for C-language programming, in addition to the database we already have for MM, then the user could switch between the two databases to get a TIPS tool specialized for either C or MM.

Learning, Doing, and Remembering a Task

This work was motivated by the need to make computer-based tasks easier to learn, do, and remember. In this section, we examine how a TIPS tool addresses these aspects of coping with a complex task.

Learning. A TIPS tool addresses learning in two ways: by enhancing learning efficiency and reducing the amount you need to learn.

To enhance learning efficiency, a TIPS tool provides the following conditions that are known to foster learning:

- Learning by doing—The TIPS tool helps you solve real problems to get real work done. Each time you use the TIPS tool to do real work you are practicing, which is indispensable for learning a complex procedure.[13] The motivation to learn is heightened because whatever is learned is put to use immediately and reinforced by work successfully completed.
- Learning from examples—To encourage learning, a TIPS tool makes examples easily available, provides explanations of unfamiliar material in the examples, and allows you to use the examples as a starting point for your own version of a similar document. Learning from examples is a form of *inductive learning*.[9]
- Learning in small amounts—With a TIPS tool, learning occurs in small chunks over a long period—weeks or months or even years—unlike a class where a large amount of material is covered in a few days. The consequent *spacing effect*[9] enhances learning.
- Review and reinforcement—A TIPS tool offers continuing and unlimited opportunities to review and reinforce learning. Training classes, on the other hand, offer little support for learning when they are over.
- Timeliness—With a TIPS tool, learning occurs as the need arises and thus is timely. A class, however, must be taken when it is scheduled and not necessarily when it is timely.
- Convenience—Like other on-line help systems, a TIPS tool makes learning experiences conveniently accessible with a few keystrokes, unlike off-line supports such as a class or manual. Because convenience increases access, learning is enhanced.
- Adaptability—A TIPS tool adapts to you, unlike a classroom (which is geared to the average student), or a manual (which is written to meet the needs of a hypothetical reader). You have the power and the responsibility to make a TIPS tool accommodate your background, needs, and desires; the kind of work you do; and your job situation.

A TIPS tool makes learning easier by reducing the amount you have to learn. You can get started with a task right away without learning a lot of information beforehand. Furthermore, the information so readily available from a TIPS tool allows you to be less than diligent about learning and committing a lot of information to memory. This is especially beneficial for the following kinds of information that are hard to remember and easy to forget.

- A TIPS tool can supply the raw forms (declarative knowledge) of the information. All you have to remember is the name of the form—better yet, the first letter alone may suffice. The TIPS tool will supply the complete form, with its mnemonically labeled arguments in their proper order.

- A TIPS tool can supply a menu that lists permissible code values for an argument. More precisely, the menu lists mnemonic descriptions of the argument values. Associated with each description is a code value, often an arbitrary number, which is the real value of the argument. Memorizing such a list is a difficult—if not futile—task, not only because of the list's nature, but also because more than one list may use the same code values.
- A TIPS tool supplies plans (procedural knowledge) that show how individual pieces have to be arranged to create a coherent whole. In $\texttt{vi}_{MM}$, plans make it possible to create entire documents without learning their formats.

Doing. To make doing a task easier, a TIPS tool does some of the work for you. In $\texttt{vi}_{MM}$, for example, if a macro argument can have a small set of possible values, the set is presented as a live menu instead of a "dead" document. When you choose an option from the menu, TIPS will substitute the corresponding value for the argument.

The plans in $\texttt{vi}_{MM}$ are another example of a TIPS tool doing some of the work. You simply identify the parts of the plan to be developed, and $\texttt{vi}_{MM}$ does much of the work of turning those parts into a document.

A TIPS tool also changes the way you go about the task. You can let the tool do some of the work, instead of doing it all yourself. Because a TIPS tool decreases effort and tedium, it has the potential to make your work easier to do, faster to finish, and less prone to errors.

Remembering. Despite your best intentions, remembering takes effort and is frustrating. Most likely, you will forget some information anyway, especially lists of hard-to-remember code values and rarely used material. But a TIPS tool's task database can "remember" for you, saving you the effort. Better yet, what a TIPS tool remembers, you don't even have to learn.

We've already talked about how a TIPS tool remembers raw forms, selectors, and plans. Another memory aid in a TIPS tool is the notebook. If you have something that TIPS doesn't provide but you might need in the future, you don't have to remember it. Instead, you can put it in the notebook and forget it. Later, you can find what you need from a menu of the notebook's contents.

Contrast this view of learning and forgetting with that taken by a class or manual. The effectiveness of a class or manual depends not only on how well you remember what you learned, but also on remembering where you learned it and where you put the manual or your notes. When you forget, you have no recourse but to search for the material, relearn it, and try not to forget again.

Does the memory service make a TIPS tool a crutch? Yes, just as a address book and a personal calendar are crutches. The extent to which a tool becomes indispensable is a measure of its worth.

Conclusion

We have demonstrated the feasibility of a software framework called TIPS for adding task-oriented, on-line services to a general-purpose base tool. The resultant specialized tool facilitates learning, doing, and remembering a complex computer-based task.

Moreover, we have demonstrated the feasibility of providing services that support *doing tasks*, in contrast to traditional forms of on-line help that support *using tools*. Mastery of a tool does not mean mastery of the tasks done with the tool—hence, the need for specialized versions of a familiar tool, such as a text editor, with services oriented toward task facilitation.

References

1. N. Gehani, *Document Formatting and Typesetting on the UNIX System*, Silicon Press, Summit, New Jersey, 1987.
2. D. Barron and M. Rees, *Text Processing and Typesetting with UNIX*, Addison-Wesley Publishing Company, Wokingham, England, 1987.
3. AT&T, *UNIX® System V Documenter's Workbench Software Release 2.0: User's Guide*, CIC No. 310-004, Issue 1, AT&T Corporate Information Center, Indianapolis, Indiana, 1986.
4. AT&T, *UNIX® System V Documenter's Workbench Software Release 2.0: Technical Discussion and Reference Manual*, CIC No. 310-005, Issue 1, AT&T Corporate Information Center, Indianapolis, Indiana, 1986.

5. C. L. Gelber and P. G. Matthews, "Teaching and Learning with UNIX Instructional Workbench™ Software," *AT&T Bell Laboratories Record*, Vol. 61, No. 10, December 1983, pp. 17-21.
6. K. Lang, R. Auld, and T. Lang, "The Goals and Methods of Computer Users," *International Journal of Man-Machine Studies*, Vol. 17, 1982, pp. 375-399.
7. S. Dutke and W. Schönpflug, "When the Introductory Period is Over: Learning while Doing One's Job," *Psychological Issues of Human Computer Interaction in the Work Place*, M. Frese, E. Ulich, and W. Dzida (eds.), Elsevier Science Publishers B.V., North-Holland, 1987, pp. 295-310.
8. J. M. Carroll and S. A. Mazur, "LisaLearning," *Computer*, Vol. 19, No. 11, November 1986, pp. 35-49.
9. J. R. Anderson, *Cognitive Psychology and Its Implications*, W. H. Freeman, San Francisco, 1980.
10. R. M. Gagné, *The Conditions of Learning*, 4th edition, Holt, Rinehart and Winston, New York, 1985.
11. K. J. Anderson, R. P. Beck, and T. E. Buonanno, "Reuse of Software Modules," *AT&T Technical Journal*, Vol. 67, No. 4, July/August 1988, pp. 71-76.
12. W. Judd, "Beyond the Menu Screen: What a Help System Can Be," *Data Training*, Vol. 5, No. 8, July 1986, pp. 24-27.
13. J. M. Carroll, R. L. Mack, C. H. Lewis, N. L. Grischkowsky, and S. R. Robertson, "Exploring Exploring a Word Processor," *Human-Computer Interaction*, Vol. 1, 1985, pp. 283-307.

(Manuscript received July 18, 1989)