(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
27 December 2001 (27.12.2001)

PCT

(10) International Publication Number
WO 01/98888 A2

(51) International Patent Classification[7]: G06F 9/00

(21) International Application Number: PCT/US01/15581

(22) International Filing Date: 14 May 2001 (14.05.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/599,086 21 June 2000 (21.06.2000) US

(71) Applicant: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, WA 98052 (US).

(72) Inventors: **ROCKEY, Eric, S.**; 622 34th Avenue, Seattle, WA 98122 (US). **TALBOTT, Shannon, P.**; 9739 112th Avenue NE, Kirkland, WA 98033 (US). **KELLY, Gavin, M.**; 2711 1st Avenue N, Seattle, WA 98109 (US). **JACOBS, Nancy, E.**; 15907 NE 113th Court, Redmond, WA 98052 (US). **HOPCROFT, Michael, J.**; 1814 3rd Street, Kirkland, WA 98033 (US). **WESTREICH, Daniel, J.**; 2106 N. 57th Street, Seattle, WA 98103 (US). **PERLOW, Jonathan, D.**; 501 Summit Avenue E, Seattle, WA 98102 (US). **ERICKSON, Paul, R.**; 24128 SE 1st Court, Sammamish, WA 98074 (US).

(74) Agents: **SADLER, Lance, R.** et al.; Suite 500, 421 W. Riverside Avenue, Spokane, WA 99201 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 01/98888 A2

(54) Title: TASK-SENSITIVE METHODS AND SYSTEMS FOR DISPLAYING COMMAND SETS

(57) Abstract: Methods and systems present commands to a user within a software application program by determining the user's context within the applicaton program and automatically presenting in a user interface commands that pertain to the user's current context. When the user's context changes, the context-sensitive commands are automatically removed from the user interface. In one implementation context blocks and context panes are employed to present the commands.

TASK-SENSITIVE METHODS AND SYSTEMS FOR DISPLAYING COMMAND SETS

TECHNICAL FIELD

This invention relates generally to methods and systems that expose commands in software application programs.

BACKGROUND OF THE INVENTION

Typically, application programs contain command sets that include individual commands that can be used by a user when working in a particular application program. These commands are specific to the purpose of the application program. For example, a word processing application program will typically include a command set that can be used to manipulate the text and/or format of a document. These command sets, however, are not always as easy to use as one would like. This situation can be complicated when a user is not familiar with the command set of an application program that they are currently using.

Current problems with application program command sets include that they can be difficult to use or browse because of the large number of commands that can be included in a command set, and that they often times can temporarily obscure a document when a user attempts to use them. In addition, command sets are typically presented in a manner that is not related to the tasks in which the user might be engaged.

With respect to the browsing difficulty of command sets, consider the following. Command sets can typically contain many different commands that are available for use. Since, in a typical user display, there is only a limited amount of

space to present information without undesirably obscuring a work area, it logically follows that not all commands can be displayed at all times for the user. To address this problem, solutions have included providing a static tool bar that can expose some commands and contain a menu structure that can be browsed by the user. Consider, for example, Fig. 1 which shows an exemplary user display 10 that includes a tool bar 12 that includes a menu structure 14 and a collection of commands 16. The menu structure 14 includes individual entries, e.g. "File", "Edit", "View", "Insert", etc. Each of these entries is associated with a drop down menu that contains a collection of individual commands that are logically related to their entry. For example, for the "File" entry, individual drop down menu-accessible commands include "new", "open", "close", "print", "print preview" and additional commands that are accessible via an "options" selection that further extends the drop down menu to display the additional commands. Each of the top line menu entries in the menu structure 14 can be associated with a drop down menu. Also, some of the commands are gathered into broad groups (such as the Font Formatting dialog) and the user needs to know what Font Formatting is, in order to find the commands in this group. Needless to say, the number of available commands can often times be quite numerous so that browsing through them is not an easy task. In addition, an inherent inefficiency with this approach is that many if not most of the displayed commands will have little or nothing to do with what the user is currently looking for. Often, in fact, many of the commands are grayed out and disabled because they are not relevant to the current task. Regardless, they are exposed to the full subset of commands.

Consider how this problem is exacerbated when a user is only moderately familiar or not familiar at all with an application program. Having to figure out and navigate through an extensive set of commands that are not explained (except

for perhaps a "Help" dialog) can make the user's experience difficult and time consuming. In addition, many application programs are physically or logically tapped out as far as including additional commands. Specifically, in many application programs there are simply so many commands in the command set that including more commands would require the menu structure to include more "options" buttons that, in turn, would present more and more commands, thus requiring a user to physically navigate through more commands.

Additionally, many application programs display their command sets in a manner that can obscure the work area for the user. For example, consider Fig. 2 which shows a "Font" dialog box 18 that is presented in the middle of the user's work area. To get to this dialog box, the user had to click on the "Format" menu entry in menu structure 14, and then select the "Font" command within the drop down menu that was presented. As a result the illustrated dialog box 18 is displayed. Although this is helpful for enabling a user to see an extensive list of commands, it is not optimal for a couple of different reasons. First, the user's document is partially obscured by the dialog box 18. This is undesirable because the user may wish to keep the work area in view. In addition, in order to work within the dialog box, the user has to quit working within their document. Thus, the dialog box is referred to as having "mode" or being "modal", meaning that a user must enter into a particular mode that renders them unable to work within their document in order to work within the dialog box. Second, and perhaps more important, the user's command selection is not implemented immediately, and, even if it were, the document is obscured by the dialog box. Specifically, in order to have a command implemented, e.g. a "strikethrough" command, the user must select text in the document that they are working on and pull up the dialog box. Only after they click on the "strikethrough" box and on the "OK" box is the

command implemented (this is somewhat related to the mode aspect mentioned above). In addition, if a user desires to implement a command on multiple different portions of text, they must separately select each text portion and apply the command for each selected portion of text. That is, for each portion of text, they must separately and individually pull up the appropriate dialog box to apply the command. This is not optimal.

Consider also the collection of commands 16. Many application programs provide such a feature where frequently used commands are displayed in a manner in which they can be quickly clicked on by a user and applied within the context of the application program. These commands are often presented as so-called "modeless" commands (as contrasted with "modal" commands) because they can be used while still working within a document. That is, in the illustrated example, individual commands from the collection include "bold", "italics", and "underline" commands. Yet, even though the goal of displaying these frequently-used commands is directed to improving user efficiency, this attempt falls short of the mark for the following reason. Even though the commands that are displayed might be considered as those that are most frequently used, their use occurrence may constitute only a very small portion of a user session, if at all. To this extent, having the commands displayed when they are not being used is wasteful in that valuable display real estate is consumed. This is a direct manifestation of the fact that the displayed commands have nothing to do with the specific context of the user. Yes--the user might in the course of their computing session have the need to use a particular command, but until that command is specifically needed by the user, its display bears no logical relation to the user's computing context.

Accordingly, this invention arose out of concerns associated with providing improved methods and systems for presenting command sets to users.

Specifically, the invention arose out of concerns associated with providing methods and systems of presenting commands in a task-sensitive manner, which assist in using physical screen space in a more efficient manner.

SUMMARYOF THE INVENTION

Methods and systems present commands to a user within a software application program by determining the user's context within the application program and automatically presenting in a user interface context-sensitive commands that pertain to the user's current context. When the user's context changes, the context-sensitive commands are automatically added to the user interface or removed from the user interface so that space can be freed up for additional, more relevant commands.

In one implementation context blocks and context panes are employed to present the commands. The context blocks and panes are displayed in a context UI container that is located adjacent a document area in a user interface. When displayed, the context blocks and context panes do not obscure a user's document.

The context blocks and panes each comprise a title bar area that labels the block or pane, and a controls area that presents commands to the user. The commands within the controls area can be logically grouped. The context blocks contain primary commands that pertain to a user's present context. More than one context block can be displayed in the context UI container. Context panes are associated with a context block and are not automatically displayed. Rather, a user can select one or more context panes from a particular context block, keyboard shortcut, or from other tool bars. The context panes contain additional commands that are logically associated with the commands of their associated

context block. When displayed, the context panes replace the context blocks in the context UI container and must be closed by the user.

In one embodiment, a user's context is determined by monitoring the user's action within a particular application program. A series of expressions are provided and describe conditions that are associated with aspects of a user's interaction with the application program. Each expression is associated with a context block. As the user's context changes, at least portions of the expressions are evaluated to determine whether its associated context block should be displayed. In one optimization, each expression is represented as a tree structure with a root node and multiple nodes associated with the root node. Each of the nodes has a value associated with it that can change as a user's context changes. When a user's context changes, individual node values are evaluated for a change. If a node value changes, then its parent node is notified with the change. The parent node then evaluates its value to ascertain whether it has changed responsive to its child's value change. This process continues whenever a node value changes. If the root node value changes, then the context block with which it is associated is either automatically displayed or removed.

In one particular advantageous implementation, a single application program is provided with a single navigable window. The application program comprises multiple different functionalities to which the single navigable window can be navigated. The different functionalities enable a user to accomplish different tasks. For example, in one implementation, a user might read electronic mail, compose an electronic mail message, or navigate to sites on the web. As the user navigates between functionalities and as their context changes within particular functionalities, context blocks are automatically presented and/or removed so that they can have useful commands at hand to use. The

functionalities of the single application program are desirably extensible, e.g. by incorporating third party functionalities that can be delivered over the web, so that the application can incorporate many different types of functionalities. Each of the incorporated functionalities can come with its own collection of automatically displayable context blocks.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is an illustration of an exemplary user display in accordance with the prior art.

Fig. 2 is an illustration of an exemplary user display that includes a dialog box in accordance with the prior art.

Fig. 3 is a high level block diagram of an exemplary computer system that can be utilized to implement various inventive embodiments.

Fig. 4 is a flow diagram that describes steps in a method in accordance with one described embodiment.

Fig. 5 is a diagram of an exemplary user interface in accordance with one described embodiment.

Fig. 6 is a diagram of an exemplary context block in accordance with one described embodiment.

Fig. 7 is a diagram of an exemplary context pane in accordance with one described embodiment.

Fig. 8 is a flow diagram that describes steps in a method in accordance with one described embodiment.

Fig. 9 is a diagram of a table in accordance with one described embodiment.

Fig. 10 is a diagram of a tree structure in accordance with one described embodiment.

Fig. 11 is a flow diagram that describes steps in a method in accordance with one described embodiment.

Fig. 12 is a diagram of a user interface in accordance with one described embodiment.

Fig. 13 is a diagram of a user interface in accordance with one described embodiment that illustrates an exemplary functionality.

Fig. 14 is a diagram of a user interface in accordance with one described embodiment that illustrates an exemplary functionality.

Fig. 15 is a flow diagram that describes steps in a method in accordance with one described embodiment.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Overview

The methods and systems described below present commands to a user within a software application program by determining the user's context within the application program and automatically presenting, in a user interface, context-sensitive commands that pertain to the user's current context. When the user's context changes, the context-sensitive commands can be automatically removed from the user interface.

Exemplary Computer System

Fig. 3 shows an exemplary computer system that can be utilized to implement the embodiment described herein. Computer 130 includes one or more processors or processing units 132, a system memory 134, and a bus 136 that

couples various system components including the system memory 134 to processors 132. The bus 136 represents one or more of any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. The system memory 134 includes read only memory (ROM) 138 and random access memory (RAM) 140. A basic input/output system (BIOS) 142, containing the basic routines that help to transfer information between elements within computer 130, such as during start-up, is stored in ROM 138.

Computer 130 further includes a hard disk drive 144 for reading from and writing to a hard disk (not shown), a magnetic disk drive 146 for reading from and writing to a removable magnetic disk 148, and an optical disk drive 150 for reading from or writing to a removable optical disk 152 such as a CD ROM or other optical media. The hard disk drive 144, magnetic disk drive 146, and optical disk drive 150 are connected to the bus 136 by an SCSI interface 154 or some other appropriate interface. The drives and their associated computer-readable media provide nonvolatile storage of computer-readable instructions, data structures, program modules and other data for computer 130. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 148 and a removable optical disk 152, it should be appreciated by those skilled in the art that other types of computer-readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, random access memories (RAMs), read only memories (ROMs), and the like, may also be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk 144, magnetic disk 148, optical disk 152, ROM 138, or RAM 140, including an

operating system 158, one or more application programs 160, other program modules 162, and program data 164. A user may enter commands and information into computer 130 through input devices such as a keyboard 166 and a pointing device 168. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are connected to the processing unit 132 through an interface 170 that is coupled to the bus 136. A monitor 172 or other type of display device is also connected to the bus 136 via an interface, such as a video adapter 174. In addition to the monitor, personal computers typically include other peripheral output devices (not shown) such as speakers and printers.

Computer 130 commonly operates in a networked environment using logical connections to one or more remote computers, such as a remote computer 176. The remote computer 176 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to computer 130, although only a memory storage device 178 has been illustrated in Fig. 3. The logical connections depicted in Fig. 3 include a local area network (LAN) 180 and a wide area network (WAN) 182. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

When used in a LAN networking environment, computer 130 is connected to the local network 180 through a network interface or adapter 184. When used in a WAN networking environment, computer 130 typically includes a modem 186 or other means for establishing communications over the wide area network 182, such as the Internet. The modem 186, which may be internal or external, is connected to the bus 136 via a serial port interface 156. In a networked

environment, program modules depicted relative to the personal computer 130, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Generally, the data processors of computer 130 are programmed by means of instructions stored at different times in the various computer-readable storage media of the computer. Programs and operating systems are typically distributed, for example, on floppy disks or CD-ROMs. From there, they are installed or loaded into the secondary memory of a computer. At execution, they are loaded at least partially into the computer's primary electronic memory. The invention described herein includes these and other various types of computer-readable storage media when such media contain instructions or programs for implementing the steps described below in conjunction with a microprocessor or other data processor. The invention also includes the computer itself when programmed according to the methods and techniques described below.

For purposes of illustration, programs and other executable program components such as the operating system are illustrated herein as discrete blocks, although it is recognized that such programs and components reside at various times in different storage components of the computer, and are executed by the data processor(s) of the computer.

Context Sensitive Commands

In the described embodiment, command sets that include one or more individual commands are automatically presented to a user depending on the user's context. Specifically, depending on the type of action the user has taken, commands that are specific to that action will appear automatically thus obviating

the need for the user to hunt through a menu structure to find commands of interest. This improves upon past approaches, which always presented top level commands, even when they were not needed by the user. This is also advantageous from the standpoint of assisting users who are unfamiliar with a particular software application. In the past, these users would have to hunt through an unfamiliar menu structure to find commands that may or may not be pertinent to an action that the user desired to take. Users also had to know the names of the functionality in order to find the tools (e.g. the user needed to know what a "table" was to know that there are tools for tables in an appropriate menu). In the present case, contextually-appropriate commands are automatically presented and removed in an interface so that a user need not worry about finding appropriate commands. That is, the described embodiment maintains an invariant that contextually applicable commands are visible and other non-applicable commands are hidden from the user.

As an example, consider the following: A user is working in a word processing application and is in the process of preparing a document. The user selects, with their cursor, a portion of the text that they believe to be spelled incorrectly. Instead of having to go to a tool bar menu at the top of the document and pull down one or more other menus to find the spell checking feature, a spell checking context block automatically appears in an interface adjacent the document. The user can then correct the incorrectly spelled word using the spell checking context block. Once the word is corrected and the user's context is not longer associated with an incorrectly spelled word, the spell checking context block automatically disappears. As the user's context changes within their document, so too do the sets of automatically presented and removed commands. Consider further that the user has included a table in their document and that they

wish to manipulate the table or its contents with table specific commands. In the past, the user would typically have to pull down a table menu entry and then select from one or more commands, some of which might present a dialog box that would obscure the user's document. In the present example, a user would simply select the table by placing the cursor inside of the table to have table-specific commands that are contextually accurate and appropriate automatically displayed in a dedicated space. Thus, a user need not hunt through a large menu structure to find commands that are appropriate for use. Here, contextually proper commands are automatically presented for the user. As the user's context changes, so too do the displayed command sets.

Fig. 4 is a flow diagram that describes steps in a method in accordance with the described embodiment. The illustrated method can be implemented in any suitable hardware, software, firmware, or combination thereof. In the present example, the method is implement in software that is executing on a user's computer.

At step 400 the method starts and step 402 then determines whether the user's current context has changed. The user's current context relates to various tasks that the user is attempting to accomplish. Context can be determined from almost any detectable state in a program. In the above example, two exemplary tasks included correcting an incorrectly-spelled word and manipulating the contents of a table. The context associated with each of these tasks can be determined from such things as the type of document a user is working in (i.e. which of the multiple different functionalities the user is accessing), the state that the document is currently in, for example, whether the document is in read-only (or browse) mode or is editable, the cursor's current location within a document (i.e. is the cursor located within a table, spreadsheet, etc. within a document), or a

particular selection that might be made within the document (i.e. selection of text, a table, etc.). Other examples of things from which context can be determined include, without limitation, whether or not a certain extension has been installed, today's date is after the due date of a particular document, the document contains comments that need to be addressed, the subject line of an email message is missing, the document contains misspelled words, and the like. Any suitable way can be used to ascertain whether a user's context has changed.

If step 402 ascertains that a user's current context has changed, step 404 makes a command set change as a function of the user's context. Specifically, command sets can be added and/or removed automatically based upon the user's context.

Context Container, Context Blocks, Context Panes

Fig. 5 shows an exemplary user interface (UI) display 500 that constitutes but one implementation of a system that automatically presents context-sensitive commands to a user. Other implementations can, of course, be used without departing from the claimed subject matter.

In the illustrated example, display 500 includes a context UI container 502 and a document area 504 adjacent the context UI container. The context UI container is a narrow, vertically aligned user interface space that can be used to expose commands in a software application. The context UI container 502 is designed, in a particular implementation, to work with applications whose functionalities are primarily oriented around interacting with documents. The context UI container 502 does not preclude other UI constructs in the application, and indeed this design assumes some additional UI such as a toolbar with commands on it such as would be displayed in global toolbar area 506. Such

commands can include, without limitation, an address well for a web browser, a "Create New" button for creating a new document type, or a button that brings up "search" or "help".

The context UI container 502 is designed, in this example, to lay to the left of the document area 504 in the application. It could, however, be situated in any suitable location. The context UI container 502 contains two types of objects that are utilized to display context-sensitive commands to a user. A first type of object is a context block, exemplary ones of which are shown at 508a-c. Context blocks are essentially context-based palettes with command "shortcuts", giving the user top-level tools for the current user contexts. In the present example, there are context blocks for editing commands (block 508a), text commands (block 508b), and table commands (block 508c). The assumption in this example is that a user has a table selected, and thus all three of these blocks are relevant. A second type of object is a context pane. Context panes provide access to secondary tools for a context and are used to complete a specific task. The secondary tools are generally more specific or more advanced functionalities relating to the context block with which the pane is associated.

In the described embodiment, context blocks automatically appear and disappear based on the user's context and have two primary forms when visible: expanded and collapsed. Context blocks are expanded by default and collapse only by manual action from the user. Context panes take up the entire context UI container 502 and focus the user on completing a task before they can continue with other commands in the application.

Context UI Container

The context UI container 502 is a collapsible vertical area that contains context blocks and context panes. The context UI container 502 can toggle between expanded and collapsed states as indicated above. When collapsed, the container is not visible at all. When expanded, the context UI container 502 is typically of a fixed size, e.g. 184 pixels wide, though it can grow wider to accommodate wider context blocks or context panes.

Changing the context UI container expansion state

In the described embodiment, the user can manually change the expansion state of the context UI container by clicking on a "Tools" button on a global toolbar in an application. This button can toggle container's expansion state. When the container is expanded, the button is visualized as toggled "on". When the container is collapsed, the button is visualized as toggled "off".

In addition, the context UI container 502 can be expanded programmatically. For example, if the user clicks on a command elsewhere in the application that requires that the container to be open, then the context UI container automatically opens. When the context UI container opens, the left border of the document being viewed by the application shifts over and the document's total width is decreased by the size of the context UI container 502.

Population of the context UI container

The context UI container 502 can be populated with both context blocks or context panes. In addition, the context UI container can be populated with a help pane. The help pane looks like a context pane, but appears next to either a context pane or the context blocks; it is approximately the same size and shape as a

context pane, is launched programmatically, and is associated with a context block or context pane. Context blocks appear in the context UI container by default. Exemplary context blocks 508a-c are shown in Fig. 5. Context block 508a contains commands that are associated with editing a document; context block 508b contains commands that are associated with manipulating text within a document; and context block 508c contains commands blocks that are associated with operating on a table. As is evident, more than one context block can be displayed at one time.

Context panes, on the other hand, are configured in this example so that they can only be viewed one at a time. And, while context blocks are displayed automatically depending on the user's context, context panes are displayed, in this example, when they are selected by a user. Context panes are task oriented and are dismissed by a user when the task is completed. Although the user dismisses them, context panes are modeless in that the user can continue acting on the document while the pane is open. Context panes can also be used to communicate an alert to a user. For example, if the user is trying to publish a document, but cannot do so because of a merge conflict, a merge conflict user interface may be provided through a context pane.

To determine which context blocks to automatically display, one implementation uses an expression-based system that ascertains the user's context and then sees to it that the proper context blocks are displayed. An exemplary expression-based system is described in more detail below in a section entitled "Expression Evaluation".

Sizing and overflow issues

The context UI container 502 is of fixed size vertically, but the number and size of context blocks is not limited. Therefore, there may be situations in which an application does not have enough room to display all of the current context blocks. This is referred to as an "overflow case". The overflow case occurs when there is not enough vertical room in the context UI container 502 to display all of the context blocks. One solution of the overflow case, in this particular example, is as follows: When an overflow occurs, the application can display a small scroll button at the bottom of the context block container. This button serves as an indicator that there are one or more context blocks scrolled out of the container. Clicking on the button once scrolls down by a predetermined number of pixels, e.g. 44 pixels, or to the end of the last context block, whichever is less. This, in turn, causes part or all of some context block to scroll off of the top of the context UI container 502. Accordingly, the context UI container 502 will also show a scroll button at its top when this occurs. Clicking on this top button will scroll up by a predetermined number of pixels, e.g. 44 pixels, or to the top of the first context block, whichever is less.

If there are no more context blocks or parts of context blocks scrolled out of the container in a certain direction (either up or down), then the corresponding respective scroll button will disappear. Since the scroll buttons take up space in the container 502, the calculation for when the scroll buttons should disappear takes into account the additional room that would appear if the button were not there.

Application window resizing issues and the context UI container

The context UI container 502 is defined in the illustrated example to have a standard size horizontally and is sized to fit the application window vertically.

The context UI container 502 responds to window/resolution resizing issues as follows: Vertically, the container resizes to fit into the space left over in the application frame from any other UI areas at its top or bottom. If the vertical space is not enough to hold all context blocks, then the overflow mechanism described above is invoked. Horizontally, the context UI container does not resize except to accommodate larger context blocks or context panes. Other than this case, the container only expands or collapses completely. The container does not resize horizontally due to the application window resizing. If the user resizes the window horizontally so that the window is narrower than the context UI container, the container will be clipped.

Context Blocks

In the illustrated example, context blocks are rectangular control containers that expose top-level commands for a given context. A context is anything that can be described by an "expression" in the application. Examples of expressions are given below. Typical contexts include: the type of document being currently viewed, the state that the document is currently in, for example, whether the document is in read-only (or browse) mode or is editable, and any objects that are currently selected in the document. Other contexts were listed above. Context blocks appear and disappear based on whether an expression that describes their context is true or false, respectively. This is discussed in more detail in the "Expression Evaluation" section below.

Fig. 6 shows exemplary context block 508b apart from the context UI container 502 of Fig. 5. Context block 508b displays text formatting commands. Each block comprises a title bar area 600 and a controls area 602.

The title bar area 600 provides a location to label the entire context block, provides expand/collapse functionality, and also contains a button 604 that opens up the context block menu. The user can click anywhere on the title bar outside of the menu button to toggle the expansion state of the context block. On the right-hand side of the title bar area 600, button 604 can be clicked to bring up a menu that can contain links to context panes, as well as commands that execute immediately without invoking a context pane. The menu then closes after the user invokes a command from it.

The controls area 602 is the main area of the context block and exposes commands to the user. In the illustrated example, the controls area allows any arbitrary HTML code to be filled into it, so commands can be exposed in any way that HTML supports. Of course, other manners of defining commands other than HTML can be used. Context blocks are advantageously "modeless" meaning that any action taken by the user is immediately applied to the user's selection in the document. This is advantageous over past methods because a user can experiment with different selections that are available through the context block and see their choices immediately effectuated in their document. In some cases, this eliminates the need for a costly (in terms of both space and time) "preview pane." This does not, however, mean that the context blocks must always be modeless. For example, a context block for inserting a table could be provided where the user sets the number of rows and columns before pressing an "insert" button.

One feature of context blocks that prevents them from being inadvertently displayed is that a selection that is made by the user in a particular document (e.g. with their cursor) must contain only content that is pertinent to a particular set of displayable commands. If the user's selection includes an object but also additional content around the object, then the context block for that object will not

be displayed. So for example, in order to have a table formatting context block visible, the user's selection would need to contain only the table, or the user's insertion point (i.e. cursor) would have to be in the table.

Context Panes

In the described embodiment, context panes, like context blocks, are also rectangular controls containers. One primary difference in this implementation is that context panes are typically larger than context blocks and only one context pane can be viewed at one time. It will be appreciated, however, that this is simply a choice for this particular implementation.

Context panes are used to expose secondary commands as well as task-based UIs, such as stepping through document publishing merge conflicts, in the application. Context panes can also be used to display various error messages related to the current document, such as when there are versioning problems when publishing a document and the user needs to make a decision about which version to keep. To maintain consistency, commands that are contained in context blocks are also repeated in their associated context panes.

Context panes can be accessed from the menus of their associated context blocks, through keyboard shortcuts, or from other UIs in the application, such as from a global toolbar. In the described embodiment, when a context pane is invoked, the entire content of the context UI container 502 (Fig. 5) is replaced with the context pane. Advantageously, context panes are typically modeless with respect to the document. This means that the user can continue to interact with their document while a context pane is open. For example, if a user wishes to use a strikethrough command repeatedly in a portion of text, the user can do this time after time by simply selecting the appropriate text and clicking on a strikethrough

box in the context pane. In the past, a user would have had to traverse a menu structure for each separate invocation of the strikethrough command.

Fig. 7 shows an exemplary context pane 700 that includes secondary format font commands. Context pane 700 includes a title bar 702 and a controls area 704 that contains individual commands. A context pane looks similar to a context block that fills up the entire context UI container. There are, however, a few differences in the described example. Here, context panes are not collapsible. That is, they are displayed in their entirety for a user during the course of a user's interaction with them. In addition, context panes have a standard way to be closed, e.g. a "Close" button or the equivalent at the bottom of the panel can be clicked on by the user.

In addition, whereas a user does not have to request a context block in order for it to appear, a user does, in most cases, request a context pane for it to appear. That is, context panes are not automatically displayed as a result of an expression evaluation. Rather, they are displayed based on some event or function call. There can be, however, some context panes that are expression based and are not end-user initiated (e.g. an error context pane that informs a user of an error condition). In addition, in this example, the user must also physically dismiss a context pane when it is no longer needed. That is, the context pane will not automatically disappear. Thus, the user is free to move their selection around the document while interacting with a context pane. If the context pane's functionality is context sensitive (for instance if it contains tools that only apply to a table within the document), then the controls in the context pane become disabled if they are out of context. The context pane typically does not disappear or collapse, in many instances, when it goes out of context.

User assistance in context panes

Context panes are used to provide access to the full range of commands in an application. In this way, context panes complement context blocks. There may be instances where a user may not be familiar with all of the commands that are displayed in the context pane. Accordingly, in the described embodiment, the context panes provide context-sensitive access to help for their commands via a pop-out help pane. This help pane appears to the right of the context pane and causes the context UI container to grow horizontally. This pushes the user's document somewhat to the right. The help pane provides quick, contextual help on how to use the controls on the pane or context block. In the Fig. 7 example, the help pane is accessed by a help icon 706 ("?") on the right side of the context pane's title bar 702.

This is much different from current help features in most current application programs. Presently in many applications, a user will have to either search through a help menu to find particular topics of interest, or they may have to enter a search query through a dialog box that typically pulls up multiple topics that relate to their search. They must then navigate through the different multiple topics to find the one in which they are interested. "Help" in these instances, is typically delivered as a separate application, overlaying, obscuring, or displaying outside of the user's application window. There is no awareness of the context of the user's work. Here, however, the described approach is somewhat different. First, the help function is contextually related to the current context pane. Thus, the user only receives help information that is pertinent to their current context. Additionally, because the help information is specifically tailored to the user's current context, and because there is a dedicated space for the context blocks and context panes, more thorough help information can be displayed in the container

UI than would normally be possible in present systems. In addition, the help feature is rendered in a "modeless" fashion so that the user can continue working in their document while the help menu is displayed. Further, it is worth noting that the contextual help provided by the present example is tailored not only to the user's context, but to the tasks and troubleshooting steps that are most likely to be needed by the user in that context. For instance, if the user is correcting a misspelled word in the document by using a context pane designed for that purpose, the help pane associated with that context pane may contain information about how to correct the misspelled word with one of the provided choices, how to add the word in question to the system dictionary, and how to replace the word in question with a different word altogether. Accordingly, the user is provided with assistance in a much more efficient and informative way.

Stackability

Although only one context pane can be viewed at a time, it is possible for multiple panes to be stored in a stack. In this case, closing one context pane reveals the next context pane in the stack. There are a number of cases where context panes can get stacked on top of each other. The following constitute some exemplary cases:

- A context pane is open and then an error context pane is displayed
- A context pane is open and the user then opens up another context pane from a button on the global toolbar.
- A context pane has a button that opens another context pane.
- A context pane is open and the user hits an accelerator key that opens up another context pane

In each case above, the latter context pane opened goes to the top of the stack, while the previous context pane goes underneath it on the stack.

In addition, each document can have its own stack of context panes. If the user navigates away from a document and back to it, the document's stack of context panes persists and is redisplayed (though it is possible for members of a stack to be aged out).

Expression Evaluation

As described above, context blocks are automatically presented to the user depending on the user's current context. In the described embodiment, an expression-based method is used to ascertain which contexts blocks to present and when to present them.

One way of implementing an expression-based method is as follows. Each context block is associated with an expression that can evaluate to a predetermined value. Each expression is essentially a defined condition that describes some aspect of a user's interaction with a document. As a user interacts with a document, the expressions, or at least portions of the expressions, are evaluated to ascertain whether they evaluate to the predetermined value. When one or more of the expressions evaluates to the predetermined value, the context block that is associated with that expression is displayed for the user.

Fig. 8 is a flow diagram that describes steps in a method in accordance with the described embodiment. The described method can be implemented in any suitable hardware, software, firmware or combination thereof. In the illustrated example, the method is implemented in software.

Step 800 associates a context-sensitive UI with a visibility expression. An exemplary context-sensitive UI is a context block as described above. In the described example, a table is used for the association and includes two columns, one of which is associated with a particular context block, the other of which is

associated with the context-block's visibility expression. Fig. 9 shows an exemplary table 900 with columns 902 and 904. Column 902 contains entries associated with each context block, while column 904 contains so-called visibility expressions that are associated with each of the context blocks. In the illustrated example, two exemplary context blocks are shown in column 902 with their corresponding visibility expressions in column 904. For example, for the "Font Format" context block the visibility expression is "em & ts". The visibility expression is a Boolean expression that describes a condition in which the application is in "edit mode" (i.e. "em") with a portion of text having been selected (i.e. "ts"). For the "Table Commands" context block, the visibility expression is "em & ip=t + tbs" which translates to a condition in which the application is in edit mode and an insertion point lies within a table (i.e. "ip=t"), or a table has been selected (i.e. "tbs").

Step 802 determines whether a visibility expression has changed in value because of a user's action. A user's action can typically change their context. A user's context could be based upon any type of variable such as user selection, insertion point, time of day, user's name, to name just a few. If the value of a visibility expression has changed, then step 804 removes visible UIs (i.e. context blocks) that are not applicable to the current context. Step 806 displays UIs that previously were not visible but are applicable to the user's current context.

The visibility expressions can be evaluated in any suitable fashion. For example, each time the user takes an action within a document, all of the expressions in table 900 can be evaluated. A more desirable approach is as follows:

Each of the expressions is represented in a data structure known as a "tree". Fig. 10 shows exemplary tree structures for the two illustrated visibility

expressions of Fig. 9. Here, the top node of each tree comprises an operation. In the present case, the operation happens to be an "AND" operation for each expression. Each top node has one or more children nodes that can either be operands or operations. In the case of the font format context block expression, each of the children nodes 1002, 1004 is an operand (i.e. "edit mode" and "text selected" respectively). For the table commands context block expression, child node 1008 is an operand (i.e. "edit mode") and child node 1010 is an operation (i.e. an "OR" operation). In turn, node 1010 has two operand nodes 1012, 1014 (i.e. "insertion point = table", and "table selected" respectively).

Each tree structure can evaluate to either "TRUE" or "FALSE". If the tree structure evaluates to "TRUE", then its corresponding context block is displayed. If the tree structure evaluates to "FALSE", or remains in a false state after some of the nodes have been evaluated, the context block is not displayed. The expression value of a tree, however, cannot change unless the value of its operands changes. For example, consider the font format tree structure. Assume that its current value is "FALSE" (indicated by the "F" adjacent node 1000). Assume also that its edit mode operand 1002 has a value of "TRUE" and its text selected operand 1004 has a value of "FALSE". In this case, the user is currently in edit mode but has not selected any text. In order for this tree structure to change in value, the value of its text selected operand 1004 must change from "FALSE" to "TRUE". This will only happen when a user has selected some text with their cursor. In accordance with the described embodiment, when the value of a child node changes, it generates a notification to its parent node that its value has changed. The parent node expression is then re-evaluated to ascertain whether its value has changed. If its value has changed, then if it has a parent node, it generates a notification that is sent to its parent node. This continues until either a parent node's expression does

not change in value, or the top parent node's expression changes in value. If the latter is the case, a corresponding context block is either displayed or removed. If the former is the case, then a current state is maintained (i.e. if the context block was previously displayed, then it is still displayed; and if the context block was not previously displayed, then it is not displayed). Thus, in many cases, only a portion of the visibility expression will need to be evaluated.

As another example, consider the visibility expression for the table commands context block. Assume that the current state of the expression is as indicated in the table below:

Node	**Value**
AND	FALSE
Em	TRUE
OR	FALSE
ip=t	FALSE
tab sel	FALSE

In this example, the table commands context block is not being displayed because the top node 1006 has evaluated to "FALSE". The user is in edit mode and neither the insertion point is in a table nor has a table been selected. Assume now that the user selects a table with their cursor. In this case, the value associated with node 1014 is changed to "TRUE". Because this node changed in value, it generates a notification and sends the notification to its parent node 1010. Node 1010 is an OR expression whose value now re-evaluates to "TRUE". Because this node has changed in value, it generates a notification and sends the

notification to its parent node 1006. Node 1006 is an AND expression that now evaluates to "TRUE". Since this is the top node and it now evaluates to "TRUE", the context block with which it is associated is now displayed for the user. This logically follows from the user's actions. That is, in order to change the value of node 1014, the user had to select a table. When the user selects the table, the table commands context block should automatically be displayed for the user. If and when the user "unselects" the table, the value associated with node 1014 will change and this change will be propagated up the tree in the form of notifications until the top node 1006 is re-evaluated to "FALSE" and the context block is removed.

Fig. 11 is a flow diagram that describes steps in an exemplary expression-evaluation method in accordance with the above-described embodiment. The described method can be implemented in any suitable hardware, software, firmware, or combination thereof. In the illustrated example, the method is implemented in software.

Step 1100 represents each expression as a tree structure having multiple nodes. Exemplary expressions and tree structures are shown and described in connection with Figs. 9 and 10. Step 1102 gets the first tree or tree structure and step 1104 gets the first leaf node on the tree. In the Fig. 10 example, exemplary leaf nodes are shown at 1002, 1004 for the font format tree, and 1008, 1012, and 1014 for the table commands tree. Step 1106 evaluates the node. Step 1108 determines whether the node value has changed. If the node value has not changed, then step 1110 determines whether there are more nodes on the tree, and if so, step 1112 gets the next node and returns to step 1106. If there are no more nodes on the tree, step 1114 determines whether there are more trees. If there are additional trees, step 1116 gets the next tree and returns to step 1104 to evaluate

the nodes of the tree. If there are no additional trees, then step 1114 returns to step 1102. Note, that the return of step 1114 can take place automatically to repeat the above process, or the return can be effected by a user's context change.

If, at step 1108, the node value has changed, step 1118 determines whether this node is the root node of the tree. If the node is the root node, then the method branches to step 1126 to ascertain whether the value of the node is "TRUE" or "FALSE". If the value is "FALSE, then step 1128 hides the context block that is associated with that particular visibility expression. If, on the other hand, the value is "TRUE", then step 1130 displays the context block that is associated with that particular visibility expression. If, at step 1118, the node is not the root node, step 1120 gets the parent node of that particular node and step 1122 evaluates the parent node. If the node value changes (step 1124), then the method branches back to step 1118. If, on the other hand, the node value does not change, then the method branches to step 1110.

The above-described process is advantageous in that many times the complete expressions that are associated with the context blocks need not be evaluated. Many times, only portions of the expressions need to be evaluated. If a particular portion of the expression has changed in value, then additional portions of the expression can be evaluated. If particular portions of the expression have not changed in value, then it is possible to terminate the expression-evaluation process thereby saving processing overhead.

Note that a small delay function can be built into the system so that the expression evaluation process is not initiated immediately when a user takes a particular action. For example, the system might be programmed so that the expression evaluation process is not initiated until a user has left their cursor in a

particular location for a definable amount of time. Such delay mechanisms will be understood by those of skill in the art and are not discussed in detail any further.

Single Navigable Window Application

In accordance with one implementation, the context-sensitive context blocks and context panes can be employed in connection with a single application program having multiple different functionalities to which a user can navigate and accomplish multiple different tasks. As the user navigates to the different functionalities, their context inevitably changes. As their context changes, so too do the context blocks and context panes that are displayed for the user. An exemplary single application program with multiple different functionalities is described in the U.S. Patent Application entitled "Single Window Navigation Methods and Systems", incorporated by reference above.

In the exemplary single application program that is subject of the reference incorporated above, software provides a user interface (UI) that presents a user with a single navigable window that can be navigated from functionality to functionality by a user. The individual functionalities are desirably provided by a single application program the result of which is a highly integrated software product.

A user, through the use of various navigation instrumentalities can navigate between the functionalities and when doing so, the single window ensures that only one functionality is presented to a user at a time. In this described embodiment, one navigation instrumentality is provided in the form of a web browser-like navigation tool. The choice of a web browser-like navigation tool follows from concerns that navigation instrumentalities be of a type that is readily understood by most individuals familiar with computing environments. Thus,

when a user first encounters the inventive navigable single window concept for the first time, they do not have to learn an unfamiliar navigation concept. Another navigation instrumentality includes links to each of the multiple different functionalities. These links can be clicked on by a user and the single navigable window is automatically navigated to the selected functionality.

Fig. 12 shows but one exemplary user interface (UI) 1200 in accordance with one described embodiment. It will be appreciated that other UIs could be used to implement the inventive concepts described herein and that the illustrated UI constitutes but one way of doing so. In the illustrated example, UI 1200 includes a navigation bar 1202, one or more command areas 1204, and a display or document area 1206 that constitutes the single navigable window.

Navigation bar 1202 is located adjacent the top of display area 1206 and contains browser-like navigation buttons 1208 in the form of a "back" button, a "forward" button, a "stop" button and the like. The navigation bar can be located anywhere on the UI. Its illustrated placement, however, is similar in appearance to the placement of traditional web browsing navigation features. In addition to the navigation buttons 1208, the navigation bar 1202 also includes links 1210 to the different functionalities that can be accessed by the user. In the illustrated example, links to three exemplary functionalities (i.e. functionality 1, functionality 2, and functionality 3) are shown. These functionalities are typically different functionalities that can enable a user to complete different respective tasks. Examples of different tasks are given below in more detail. These functionalities are all provided within the context of a single application. To access a particular functionality, a user simply clicks on one of the links and a window that pertains to the selected functionality is immediately presented in the display area 1206.

Command areas 1204 are located adjacent the top and left side of the display area 1206. The command area(s) can, however, be located in any suitable location. The command areas provide commands that are both global in nature and specific to the particular context the user has selected. For example, some commands such as "search" and "help" might be considered as global in nature since they can find use in many contexts. Other commands, such as "text bold" or "reply to all" are more specific to the particular context that the user has selected. For the "text bold" command, the user's context may likely be a word processing context, while the "reply to all" command may likely be employed in an email context.

Example

As an example of the single navigable window provided by a single application consider Figs. 13 and 14.

In this example, the multiple functionalities 1210 that can be navigated by a user include a browser functionality (indicated by the home icon), a mail functionality (indicated by the letter icon), a planner functionality (indicated by the clock icon), a contacts functionality (indicated by the people icon), a documents functionality (indicated by the folder icon), and a links functionality (indicated by the world icon). These functionalities are so-called "document-centric" functionalities because they all relate in some way to a document that a user interacts with, e.g. a Web page document, an email document, a calendar document, etc.

Fig. 13 shows an example of a display that is rendered in the display area 1206 when a user clicks on the link to the browser functionality. By clicking on the link (i.e. the home icon) to the browser functionality, single application

program software executing on the user's computer executes to implement a browser functionality. In this example, the browser functionality displays the user's home page in display area 1206. Notice also that navigation buttons 1208 are provided for navigation between the different selectable functionalities. The command areas 1204 contain context blocks designated as "Favorites" and "Browsing" that include command sets with commands that are specific to the context that the user has selected. In this example, the user's context is a browsing context. Accordingly, the leftmost command area contains commands that are specific to the browsing functionality. Such commands include ones that a user would normally expect to find in a web browser. Notice also that the command area 1204 adjacent the top of display area 1206 also contains commands that are specific to the browsing context, i.e. "Add to Favorites" and an address well in which the user can type a URL of a particular destination web site. Thus, context blocks that are displayable in the leftmost command area are automatically presented to the user as the user's context changes.

Fig. 14 shows an example of a display that is rendered in the display area 1206 when the user clicks on the link to the mail functionality (i.e. the folder icon). By clicking on this link, single application program software executing on the user's computer executes to implement the mail functionality. In this example, the mail functionality displays a user's inbox with messages that have been received by the user. Notice that the leftmost command area has been minimized by the user and that command area 1204 adjacent the top of the display area 1206 contains commands that are specific to the user's current context, e.g. "New" for generating a new email message, "Reply" for replying to an email message, "Reply to All" for replying to all recipients of an email message and the like. When the user's context within this functionality changes in a way that requires

one or more context blocks to be displayed, the context blocks will be automatically displayed in the leftmost command area. For example, a user may author an email message and desire to italicize a portion of text. Upon selecting a portion of text, a text formatting context block will automatically appear for the user to use. As another example, assume that a user incorporates a table into their email message, if they then move the cursor inside of the table, the table formatting context block will automatically appear in the leftmost command area.

Although not specifically illustrated, the user could have displays for the planner, contacts, documents, and links functionalities presented in the display area 1206 by simply clicking on the links to these specific functionalities. When so displayed, context blocks that are associated with the user's context in these particular functionalities will be automatically displayed in accordance with the user's particular context. The navigation bar 1208 provides the user with the ability to navigate through these different functionalities in a browser-like manner.

It is important to note that the above example constitutes but one exemplary way in which multiple different functionalities and context blocks can be presented to a user within the construct of a navigable structure. It should be understood that the specifically illustrated functionalities (i.e. browser, mail, planner etc.) constitute specific examples of different functionalities that are capable of being incorporated into the single application program that provides the navigable window. Accordingly, other different functionalities can be employed. This aspect is discussed in more detail in the section entitled "Extensible Functionalities" below. It should also be noted that various context panes are associated with the individual context blocks that form the basis of this example. The context panes have not specifically been described in this example because they were explained above.

Fig. 15 is a flow diagram that describes steps in a method in accordance with the described embodiment. The illustrated method can be implemented in any suitable hardware, software, firmware, or combination thereof. In the illustrated example, the method is implemented in software.

Step 1500 provides a single application program with multiple different functionalities. The functionalities, as pointed out above, are advantageously different so as to enable a user to accomplish different tasks. One specific non-limiting example of different functionalities was given above in the context of document-centric functionalities that enable a user to make use of browser, mail, planner, contacts, documents, and links functionalities. Step 1500 can be implemented by configuring a computing device, such as a user's computer, with the single application program having the multiple different functionalities. This step can also be implemented by providing a software platform in the form of a generic single application shell that is extensible and adaptable to receive different extensions or software modules that embody various different functionalities, as described in various patent applications incorporated by reference above. These different extensions are then presented to the user in the context of the single application having the multiple different functionalities.

These extensions can be delivered to the platform in any suitable way and through any suitable delivery mechanism. For example, one way of delivering the various extensions or functionalities is to deliver them via a network such as an Intranet or the Internet. Regardless of the manner in which the single application is provided, step 1502 presents a user interface (UI) with a single window and links to the multiple different functionalities. The UI can also advantageously include navigation instrumentalities that enable a user to navigate between the different functionalities in a browser-like manner. Figs. 13-14 give specific

examples of an exemplary UI that can be used in accordance with the described embodiment. Step 1504 ascertains whether a user has selected a particular link to a functionality or whether the user has used one of the navigation instrumentalities to navigate to a particular functionality. If a user has not done either, the method branches back to step 1502. If, on the other hand, a user has selected a particular link or used a navigation tool to navigate to a particular functionality, step 1506 presents a functionality-specific display within the single window. That is, the single navigable window is navigated by the software to the selected functionality. Specific examples of this were given above in connection with Figs. 13 and 14 in which browsing and mail functionalities were respectively displayed within display area 1206. In connection with presenting the functionality-specific display in step 1506, step 1508 can present functionality-specific commands in a command area of the UI. This is advantageously done automatically as a user navigates from functionality to functionality. That is, as a user changes functionalities, command sets that are specific to the user's current context or functionality are automatically displayed in the command area. In connection with this step, context blocks can be automatically displayed as described above. It will also be appreciated that step 1508 includes the step of presenting various context panes in response to the user selecting them as described above. Step 1508 then branches back to step 1504 to ascertain whether the user has navigated to another functionality.

Context block and context pane persistence

In the multi-functionality application scenario, context blocks or context panes can be provided that are not specifically associated with a specific document. Rather, these context blocks and context panes remain open regardless

of the document, until the user explicitly closes them. Such context blocks and panes are referred to herein as "Application-level context blocks" and "Application-level context panes".

Application-level context blocks and Application-level context panes

Application-level context blocks are context blocks that are not removed from the UI, even when the user navigates to another document. So, instead of being associated with a particular document, they are associated with a state of the application as a whole. Application-level context panes are similar: they are context panes that stay open even when the user navigates to another document.

As an example, consider a user that desires to use a "search" function. If the user opens the search function, a context pane corresponding to the search function will populate the context UI container. The search context pane is specific to the user's particular functionality. Thus, if the user has navigated to their email inbox, the search context pane will enable them to search their inbox. If the user has navigated to a particular document, the search context pane will enable them to search that document. As long as the search context pane is not closed by the user it will navigate with them from functionality to functionality and enable them to specifically conduct searches with the individual functionalities.

Application-level context panes and stacking

Application-level context panes are implemented with special behavioral characteristics with regards to the stacking of context panes. In this example, there are two types of context panes: those with affinity to a particular document, and those with no affinity to any document. A stack of context panes that have

been opened is maintained. The stack is ordered so that the most recent pane is on the top of the stack. This stack does not contain any panes that have been explicitly closed by the user. The first pane in the stack that meets one of the following two criteria is displayed: (1) the pane has affinity to the current document, and (2) the pane has no affinity to any document. If no pane in the stack meets these criteria, then the context blocks are displayed. Note that this has the effect of hiding any pane that does not have affinity to the current document. This means that when navigation occurs, panes with affinity to the previous document are suppressed. Panes with affinity to the new document and those with no affinity to any document become candidates for display. They are considered as candidates because only the pane closest to the top of the stack is actually displayed.

Conclusion

The embodiments described above provide methods and systems that automatically present a user with commands that are specific to a task in which the user happens to be engaged. Advantageously, as the user's context changes within an application, the commands that are presented automatically change as well. In various implementations, the user can be given the opportunity to select additional context-sensitive commands for display. Overall, the methods and systems advantageously enable a user to take advantage of many different commands without requiring the user to know much about the application that they are using. In one particular implementation, a single application comprises multiple functionalities that enable a user to accomplish different tasks. These multiple functionalities are presented in the context of a single window that is navigable by a user between the different functionalities. Advantageously, navigation

instrumentalities are provided that are, in some instances, browser-like in appearance and allow the user to navigate between the application-provided functionalities in a browser-like manner. Functionality-specific commands can be automatically presented to the user when they navigate to a particular functionality. The functionality-specific commands are presented, in the illustrated example, in the form of context blocks and content panes as described above. One aspect of the single navigable window application is that the application can serve as a basis for an extensible platform that can be configured with different functionalities. Software modules incorporating the different functionalities, as well as appropriate command sets that are displayable in the context blocks and panes, can be desirably included in the software modules. When the modules are plugged into the platform, a set of extensible functions is provided. Each of the extensible functions can have their own set of unique context blocks and panes that can be automatically displayed in a manner that is defined by the software developer of the module.

Although the invention has been described in language specific to structural features and/or methodological steps, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or steps described. Rather, the specific features and steps are disclosed as preferred forms of implementing the claimed invention.

CLAIMS

1. A method of exposing commands in a software application program comprising:

determining a user's context within an application program; and

automatically displaying at least one command on a display for the user based on the user's context.

2. The method of claim 1 further comprising automatically removing said at least one command from the display responsive to a change in the user's context.

3. The method of claim 1, wherein the application program comprises a document-centric application program and said displaying does not obscure a document in which the user is working.

4. The method of claim 1, wherein the application program comprises a document-centric application program and said at least one command is displayed in a modeless fashion in which the user can continue to work within a document while said at least one command is displayed.

5. The method of claim 1 further comprising after said displaying, executing a command without requiring any action from a user other than selecting the command.

6. The method of claim 1, wherein said determining comprises ascertaining a position of a user's cursor within a document provided by the application program.

7. The method of claim 1, wherein said determining comprises ascertaining a user's selection within a document provided by the application program.

8. The method of claim 1, wherein said determining comprises one or more of:

ascertaining a position of a user's cursor within a document provided by the application program; and

ascertaining a user's selection within a document provided by the application program.

9. The method of claim 1, wherein said context pertains to various tasks the user may attempt to accomplish.

10. The method of claim 1, wherein said context pertains to one or more of the following: a type of document the user is working in, a state of a document the user is working in, a cursor's location within a document, and a particular user selection within a document.

11. The method of claim 1, wherein said displaying is independent of a user selecting any displayed menu item.

12. One or more computer-readable media having computer-readable instructions thereon which, when executed by a computer, cause the computer to:

determine a user's context within an application program;

automatically display at least one command on a display for the user based on the user's context, said at least one command being displayed in a modeless fashion in which the user can continue to work within a document provided by the application program while said at least one command is displayed; and

automatically remove said at least one command from the user's display responsive to a change in the user's context.

13. The computer-readable media of claim 12, wherein the computer determines the user's context by one or more of the following:

ascertaining a position of a user's cursor within a document provided by the application program; and

ascertaining a user's selection within a document provided by the application program.

14. A method of exposing commands in a software application program comprising:

displaying a user interface comprising:

a document area in which a user can work on a document that is provided by an application program; and

a container area proximate the document area; and

automatically presenting one or more command sets in the container area based upon a task that the user is attempting to accomplish on a document in the document area.

15. The method of claim 14, wherein individual command sets are defined by context blocks, at least one context block containing multiple commands that are possible selections for the task that the user is attempting to accomplish.

16. The method of claim 15, wherein the context blocks are collapsible within the user interface.

17. The method of claim 15 further comprising:

receiving user input; and

responsive to receiving said input, displaying additional commands in the container area.

18. The method of claim 17, wherein said displaying of the additional commands comprises replacing any context blocks in the container area with the additional commands.

19. The method of claim 17, wherein the additional commands are displayed within a user-closeable context pane.

20. The method of claim 19, wherein the context pane is configured to remain open within the container area until it is closed by the user.

21. The method of claim 19, wherein the context pane fills the container area.

22. The method of claim 14 further comprising automatically removing said one or more command sets without requiring user intervention.

23. The method of claim 14 further comprising automatically removing said one or more command sets when the user is no longer attempting to accomplish the task.

24. The method of claim 14, wherein the container area has an expanded state and a contracted state.

25. The method of claim 24 further comprising programmatically expanding the container area from the collapsed state to automatically display a command set.

26. One or more computer-readable media having computer-readable instructions thereon which, when executed by a computer, cause the computer to:

display a user interface comprising:

a document area in which a user can work on a document that is provided by an application program; and

a container area proximate the document area;

automatically present one or more command sets in the container area based upon a task that the user is attempting to accomplish on a document in the document area, individual command sets being defined by context blocks, at least

one context block containing multiple commands that are possible selections for the task that the user is attempting to accomplish; and

automatically remove one or more context blocks when the user is no longer attempting to accomplish the task.

27. Software code embodied on a computer-readable medium which, when executed by a computer, provides a user interface (UI) comprising:

a document area within which a user can work on a document that is provided by an application program; and

a container area proximate the document area configured to automatically display commands for a user based on the user's current context without obscuring a document that a user is working on within the document area.

28. The software code of claim 27, wherein the commands are displayed in context blocks, individual context blocks containing multiple commands that are possible selections for a user based upon their context.

29. The software code of claim 28, wherein the commands can further be displayed in context panes, each context pane being associated with a context block and being selectable for display by a user from its associated context block.

30. The software code of claim 28, wherein the context blocks are stackable within the container area.

31. The software code of claim 28, wherein individual context panes replace context blocks within the container area.

32. The software code of claim 31, wherein context panes must be closed by a user when the user no longer wishes to access commands within the context pane.

33. The software code of claim 28, wherein the context blocks are configured so that they can be scrolled through by a user.

34. A computer system comprising the user interface of claim 27.

35. A method of exposing commands in a software application program comprising:

determining a user's context within an application program; and

automatically displaying at least one context block on a display for the user based on the user's context, individual context blocks containing multiple commands that are possible selections for a user based upon their context.

36. The method of claim 35, wherein said determining comprises evaluating at least portions of one or more expressions, each expression being associated with a context block and defining a condition that describes one or more aspects of a user's interaction with the application program.

37. The method of claim 36, wherein the expressions evaluate to Boolean values.

38. The method of claim 35, wherein a user's context can be affected by one or more of the following: a document type, a document state, and objects within a document that can be selected by the user.

39. The method of claim 35, wherein said displaying comprises displaying a context block having a title bar area that labels the context block.

40. The method of claim 39, wherein the title bar area is configured to enable the context block to be toggled between expanded and collapsed states.

41. The method of claim 39, wherein the title bar area comprises a menu display button that is configured to enable a menu that is associated with the context block to be displayed.

42. The method of claim 41, wherein the menu display button is associated with a menu that contains links to one or more context panes, each context pane comprising additional context-sensitive commands.

43. The method of claim 35, wherein said displaying comprises displaying a context block with a controls area that exposes the multiple commands to the user.

44. The method of claim 43, wherein a command display within the controls area is defined in HTML.

45. The method of claim 35, wherein said displaying comprises displaying said at least one context block in a modeless fashion.

46. A method of exposing commands in a software application program comprising:

determining a user's context within an application program;

based on the user's context, displaying commands that are associated with the context and which can assist the user in accomplishing a task; and

while the commands are being displayed, enabling the user to select and apply various commands multiple times.

47. The method of claim 46 further comprising applying one or more selected commands, when selected by a user, without further user interaction.

48. The method of claim 46, wherein said displaying comprises displaying the commands responsive to the user selecting from a menu that is supported by an automatically-appearing and disappearing context block that contains context-sensitive commands.

49. The method of claim 46, wherein said displaying comprises displaying the commands in a modeless manner.

50. The method of claim 46, wherein said displaying comprises displaying the commands within a context pane having a title bar that labels the context pane and a controls area that exposes the commands to the user.

51. The method of claim 50, wherein the context pane is not collapsible.

52. The method of claim 50, wherein the context pane must be closed by the user.

53. The method of claim 50, wherein the user must request the context pane to be displayed.

54. The method of claim 50, wherein some of the commands in the controls area can be context-sensitive and are disabled if they are out of context.

55. The method of claim 50, wherein the context pane includes a context-sensitive help feature that displays help information that is contextually related to a context pane.

56. The method of claim 55, wherein the help feature is accessible via an icon on the title bar.

57. The method of claim 55, wherein the help feature is displayed in a modeless manner.

58. The method of claim 50, wherein multiple context panes are stackable in a queue.

59. One or more computer-readable media having computer-readable instructions thereon which, when executed by a computer, implement the method of claim 46.

60. A method of exposing commands in a software application program comprising:

associating multiple context blocks with individual expressions, individual context blocks containing one or more commands, individual expressions defining conditions that describe one or more aspects of a user's interaction with the application program;

evaluating at least portions of at least some of the expressions responsive to a change in the user's context; and

displaying, for the user, at least one context block responsive to said evaluating.

61. The method of claim 60, wherein the expressions define conditions that describe one or more aspects of a user's interaction with a document.

62. The method of claim 60, wherein said associating comprises maintaining a table with entries for the context blocks and their associated expressions.

63. The method of claim 60, wherein the expressions are Boolean expressions.

64. The method of claim 60, wherein said evaluating comprises:

representing each expression as a tree structure having multiple nodes, each node comprising either an expression operand or an operation;

associating a value with each of the nodes, the node values being capable of changing when a user's context changes;

for some of the tree structures, evaluating values associated with less than all of the nodes to ascertain whether to display a context block associated with the tree structure.

65. The method of claim 60, wherein said evaluating comprises:

representing each expression as a tree structure having multiple nodes, each node comprising either an expression operand or an operation;

associating a value with each of the nodes, the node values being capable of changing when a user's context changes;

if a value for a particular node changes, and if the particular node has a parent node, notifying the parent node with the changed value and re-evaluating the parent node's value.

66. The method of claim 65, wherein each tree structure has a root node, and further comprising displaying the context block associated with a tree structure only if the root node changes in value.

67. One or more computer-readable media having computer-readable instructions thereon which, when executed by a computer, cause the computer to:

associate multiple context blocks with individual expressions, individual context blocks containing one or more commands, individual expressions defining conditions that describe one or more aspects of a user's interaction with the application program;

evaluate at least portions of at least some of the expressions responsive to a change in the user's context by:

representing each expression as a tree structure having multiple nodes, each node comprising either an expression operand or an operation, each tree structure having a root node;

associating a value with each of the nodes, the node values being capable of changing when a user's context changes; and

if a value for a particular node changes, and if the particular node has a parent node, notifying the parent node with the changed value and re-evaluating the parent node's value; and

display, for the user, at least one context block responsive to said evaluating only if the root node of the tree structure that is associated with the context block changes in value.

68. A data structure embodied on a computer-readable medium and configured for use in exposing commands in a software application program, the data structure comprising:

a root node having a value; and

multiple nodes associated with the root node each of which having a value, the root node and multiple nodes being arranged to defined various parent/child

relationships and collectively representing an expression that defines a condition that describes one or more aspects of a user's interaction with the application program;

each of the multiple nodes representing either an operation or an operand, individual multiple nodes being configured in a manner such that:

a change in value generates a notification to the node's parent; and

a notification of a change in value from a child node causes a re-evaluation of the node's value;

the root node being configured such that a change in its value causes either the automatic display or removal of a set of context-sensitive commands that are appropriate for the user's context within the application program.

69. A computing system comprising:

a single application program configured to provide:

a single navigable window;

multiple different functionalities to which the single navigable window can be navigated by a user; and

at least one context-sensitive command area that is associated with the single navigable window, the single application program being configured to automatically change command sets that are presented to the user within the command area as the user navigates to different functionalities.

70. The computing system of claim 69, wherein the single application program is configured to provide navigation instrumentalities associated with the single navigable window, the navigation instrumentalities being configured for use by the user to navigate the single window to the different functionalities.

71. The computing system of claim 70, wherein one of the navigation instrumentalities comprises links associated with each of the multiple different functionalities to which the single navigable window can be navigated.

72. The computing system of claim 70, wherein one of the navigation instrumentalities comprises browser-like navigation buttons that can be used, in connection with the navigation model, to navigate the single navigable window between the different functionalities.

73. The computing system of claim 69, wherein the multiple different functionalities comprise document-centric functionalities.

74. A computing system comprising:

a single application program configured to:

display a single navigable window for a user to use in navigating between multiple different functionalities that can be provided by the single application program;

provide at least one context-sensitive command area that is associated with the single navigable window, the single application program automatically changing command sets that are presented to the

user within the command area as the user navigates to different functionalities; and

incorporate different functionalities in an extensible manner so that the user can use the single navigable window to navigate to the different incorporated functionalities.

75. The computing system of claim 74, wherein the single application program is configured to provide navigation instrumentalities associated with the single navigable window, the navigation instrumentalities being configured for use by the user to navigate the single window to the different functionalities.

76. The computing system of claim 75, wherein one of the navigation instrumentalities comprises links associated with each of the multiple different functionalities to which the single navigable window can be navigated.

77. The computing system of claim 75, wherein one of the navigation instrumentalities comprises browser-like navigation buttons that can be used to navigate the single navigable window between different functionalities.

78. A computing method comprising:

displaying a user interface that comprises a single navigable window that can be navigated between multiple different functionalities that are provided by a single application program;

receiving user input that indicates selection of a particular functionality;

responsive to receiving said user input, navigating the single navigable window to the particular selected functionality and displaying in said window

indicia of said functionality that can enable a user to accomplish a task associated with the particular selected functionality;

determining a user's context within the selected functionality; and

automatically displaying at least one command for the user based on the user's context.

79. The computing method of claim 78 further comprising automatically removing said at least one command from the display responsive to change in the user's context.

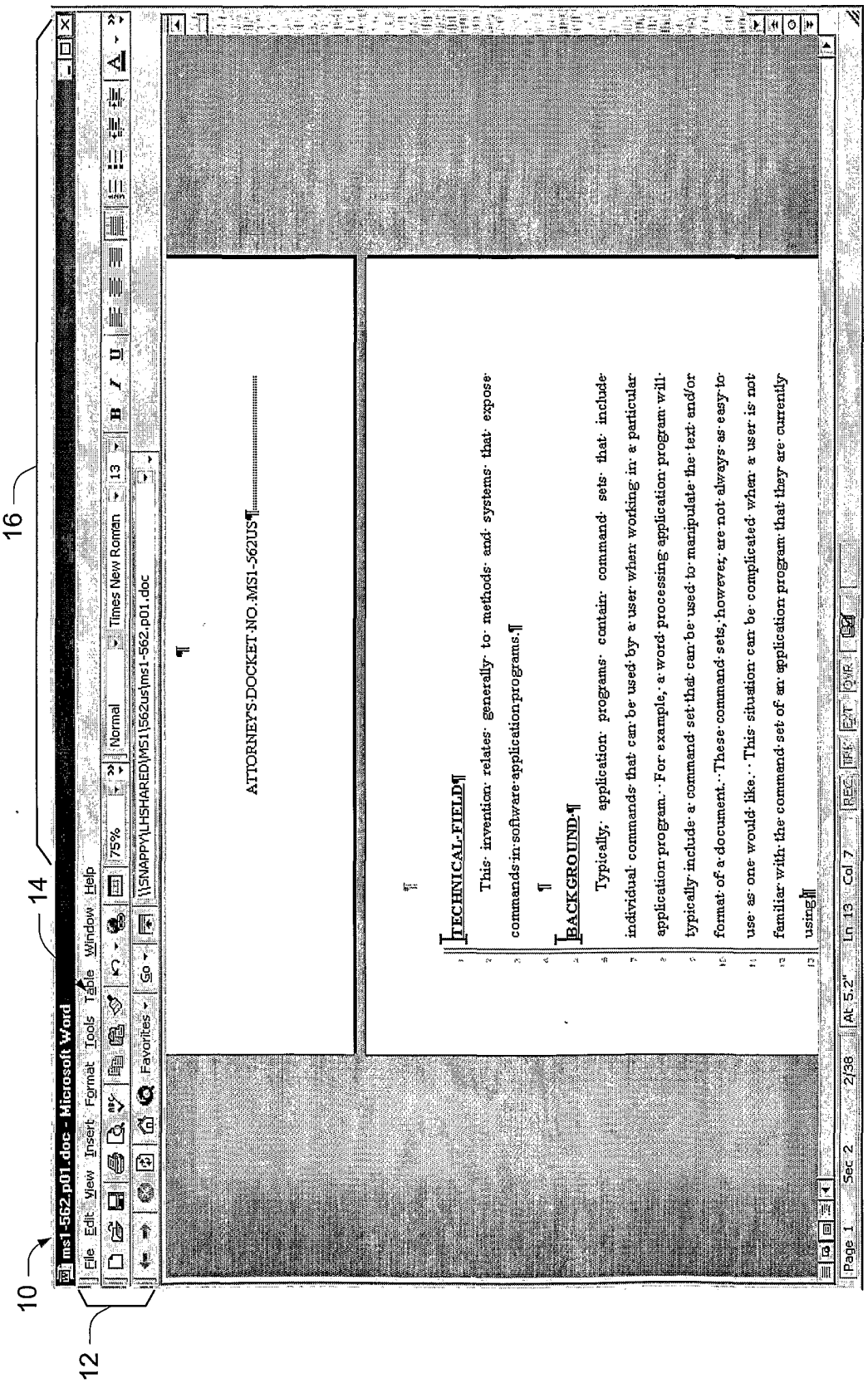

10
12
14
16
ms1-562.p01.doc - Microsoft Word
File Edit View Insert Format Tools Table Window Help
Normal
Times New Roman
13
75%
Favorites
Go
\\SNAPPY\LHSHARED\MS1\562us\ms1-562.p01.doc
ATTORNEYS DOCKET NO. MS1-562US
TECHNICAL FIELD
This invention relates generally to methods and systems that expose commands in software application programs.
BACKGROUND
Typically, application programs contain command sets that include individual commands that can be used by a user when working in a particular application program. For example, a word processing application program will typically include a command set that can be used to manipulate the text and/or format of a document. These command sets, however, are not always as easy to use as one would like. This situation can be complicated when a user is not familiar with the command set of an application program that they are currently using
Page 1
Sec 2
2/38
At 5.2"
Ln 13
Col 7
REC
TRK
EXT
OVR

Fig. 1

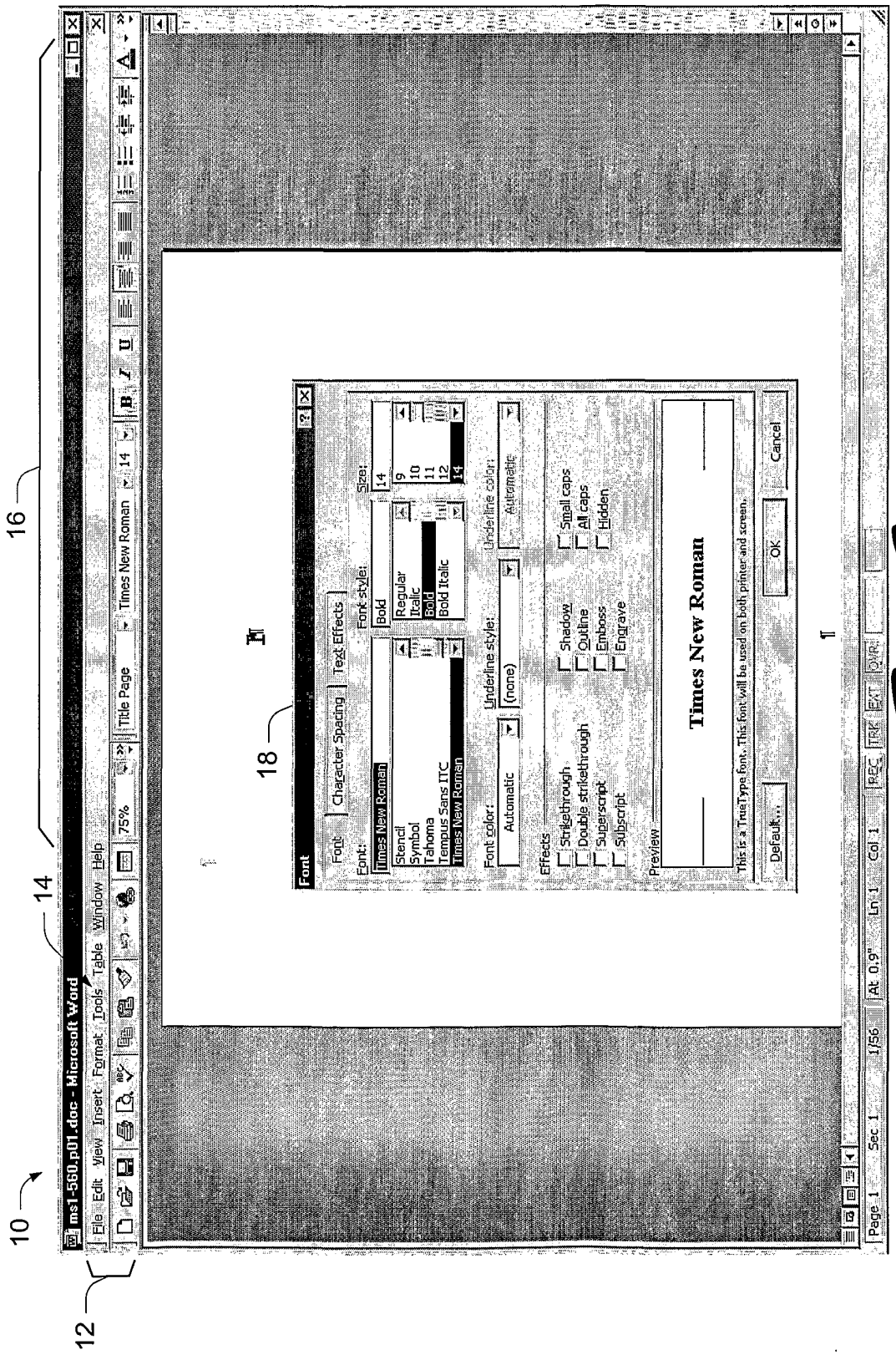
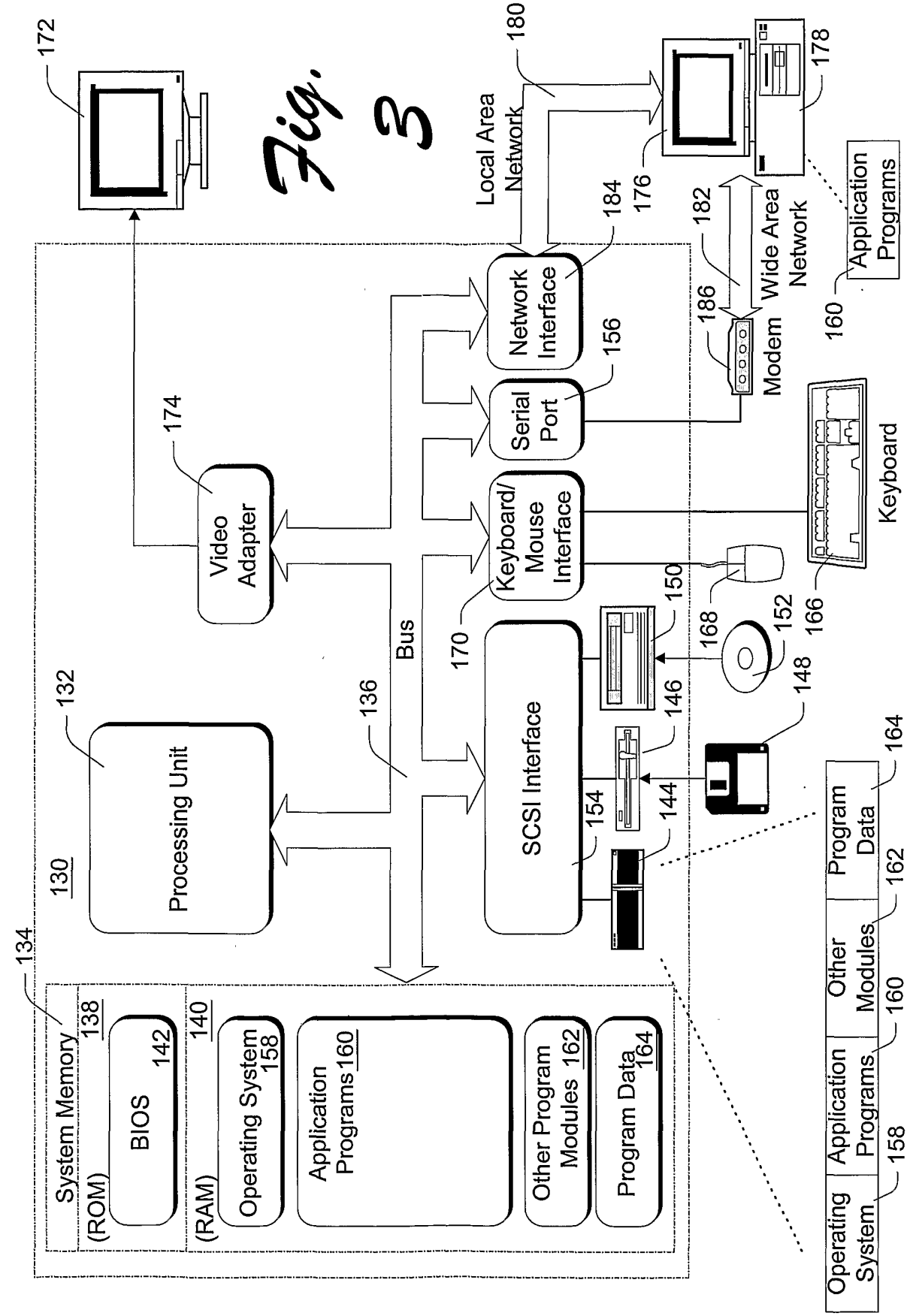

10
12
14
16
18
ms1-560.p01.doc - Microsoft Word
Font
Times New Roman
Times New Roman

Fig. 2


172
Fig. 3
Local Area Network
180
178
Application Programs
160
182
Wide Area Network
184
176
186
Modem
Network Interface
156
Serial Port
174
Video Adapter
Keyboard/ Mouse Interface
Keyboard
150
168
152
166
170
Bus
148
132
136
146
Processing Unit
SCSI Interface
154
144
130
164
Program Data
162
Other Modules
160
Application Programs
158
Operating System
134
System Memory
138
(ROM)
BIOS
142
140
(RAM)
Operating System
158
Application Programs 160
Other Program Modules 162
Program Data
164

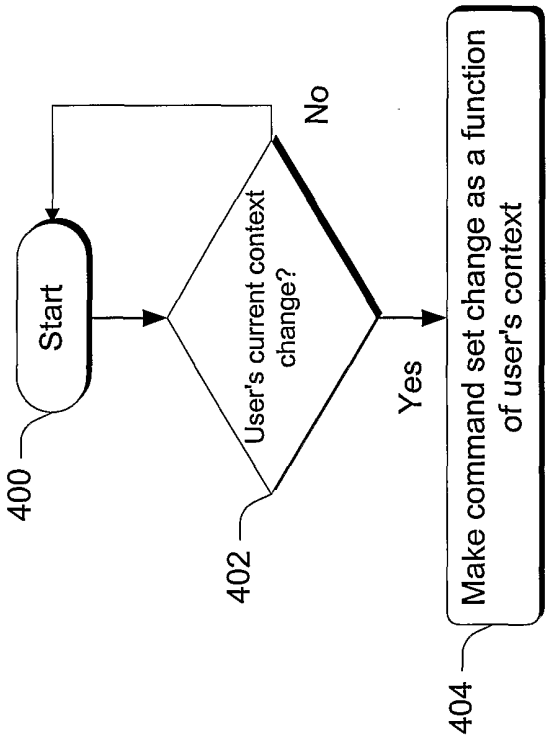

Start
400
User's current context change?
402
No
Yes
Make command set change as a function of user's context
404

Fig. 4

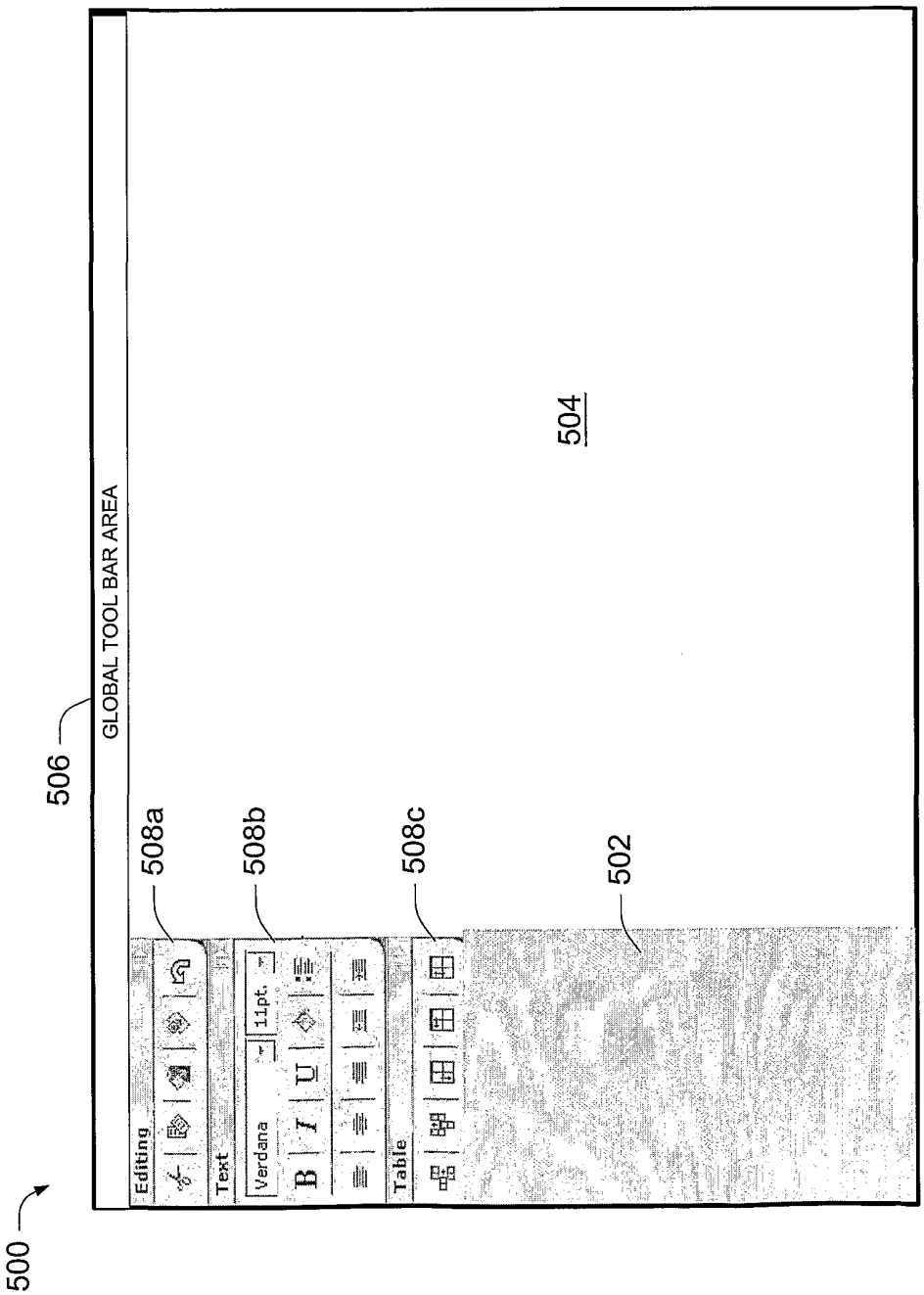

500
506
GLOBAL TOOL BAR AREA
508a
508b
508c
502
504
Editing
Text
Table
Verdana
11pt.

Fig. 5

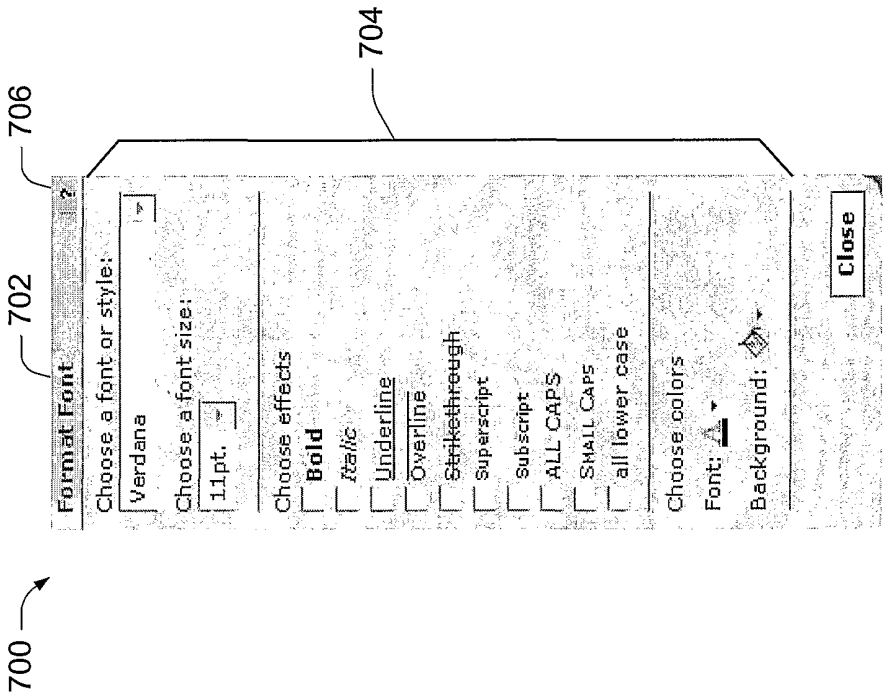

700
702
704
706
Format Font
Choose a font or style:
Verdana
Choose a font size:
11pt.
Choose effects
Bold
Italic
Underline
Overline
Strikethrough
Superscript
Subscript
ALL CAPS
SMALL CAPS
all lower case
Choose colors
Font:
Background:
Close

Fig. 7

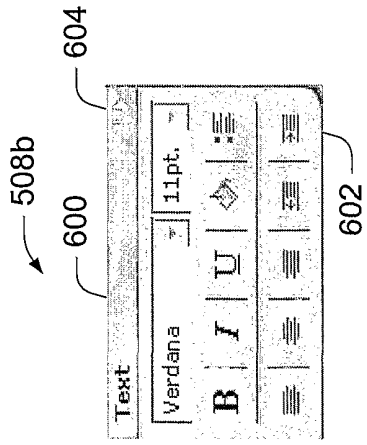

508b
600
602
604
Text
Verdana
11pt.

Fig. 6

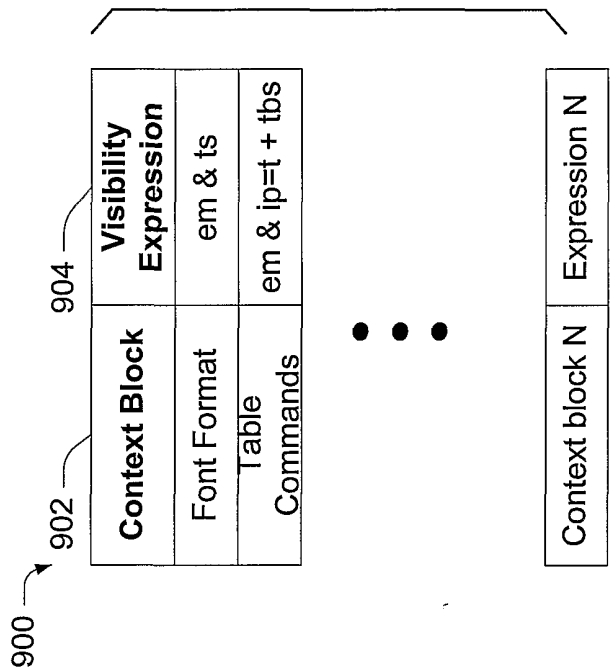

900
902
904
Context Block
Font Format
Table Commands
Visibility Expression
em & ts
em & ip=t + tbs
Context block N
Expression N

Fig. 9

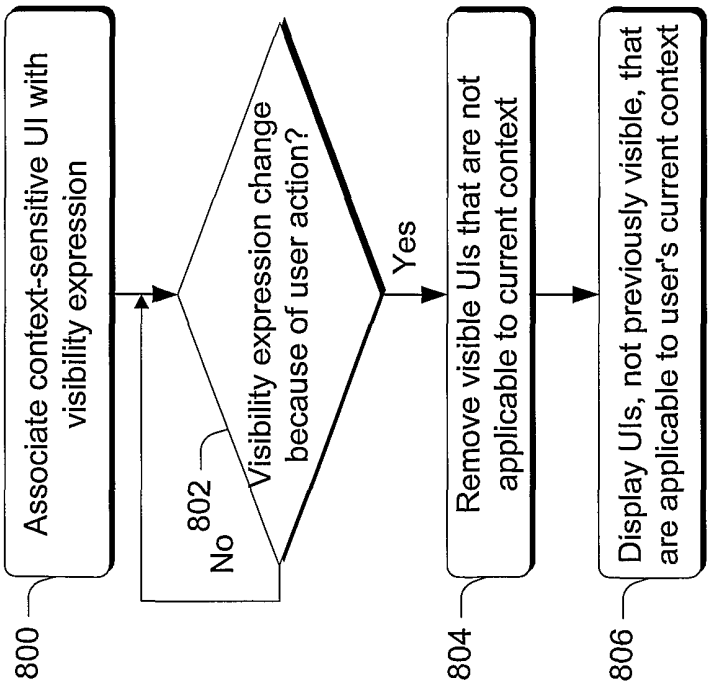

800
Associate context-sensitive UI with visibility expression
802
Visibility expression change because of user action?
No
Yes
804
Remove visible UIs that are not applicable to current context
806
Display UIs, not previously visible, that are applicable to user's current context

Fig. 8

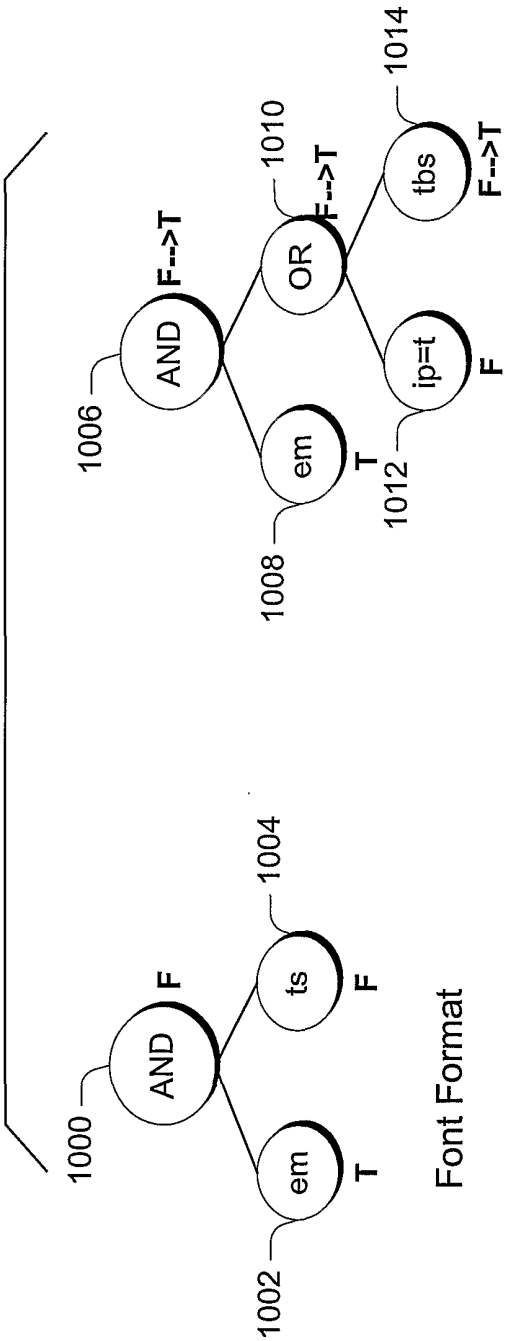

1000
AND
F
em
T
1002
ts
F
1004
Font Format
1006
AND
F-->T
em
T
1008
OR
F-->T
ip=t
F
1012
tbs
F-->T
1014
1010
Table
Commands

Fig. 10

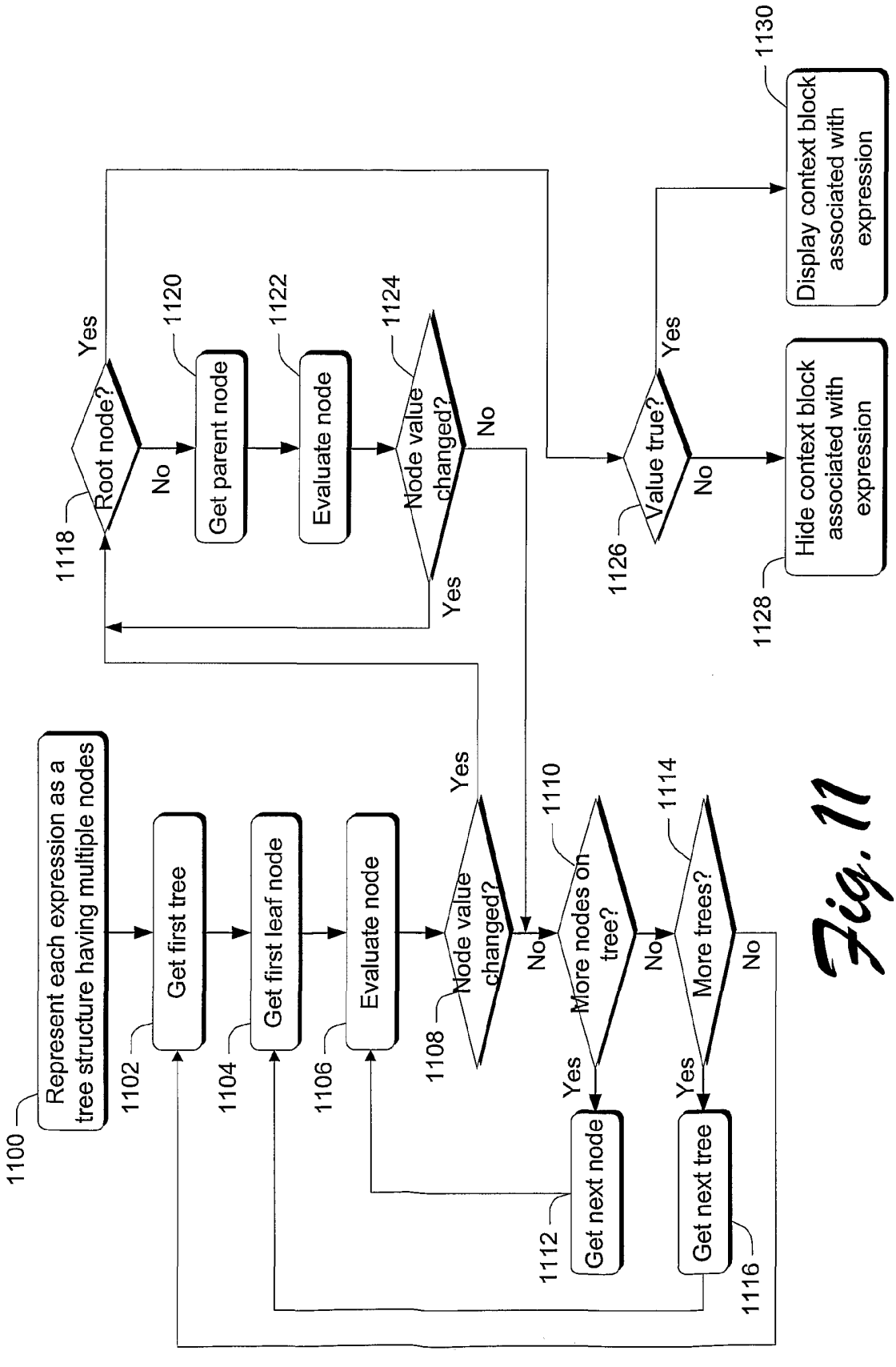

Represent each expression as a tree structure having multiple nodes
1100
Get first tree
1102
Get first leaf node
1104
Evaluate node
1106
Node value changed?
1108
More nodes on tree?
1110
Get next node
1112
More trees?
1114
Get next tree
1116
Root node?
1118
Get parent node
1120
Evaluate node
1122
Node value changed?
1124
Value true?
1126
Hide context block associated with expression
1128
Display context block associated with expression
1130
Yes
No

Fig. 11

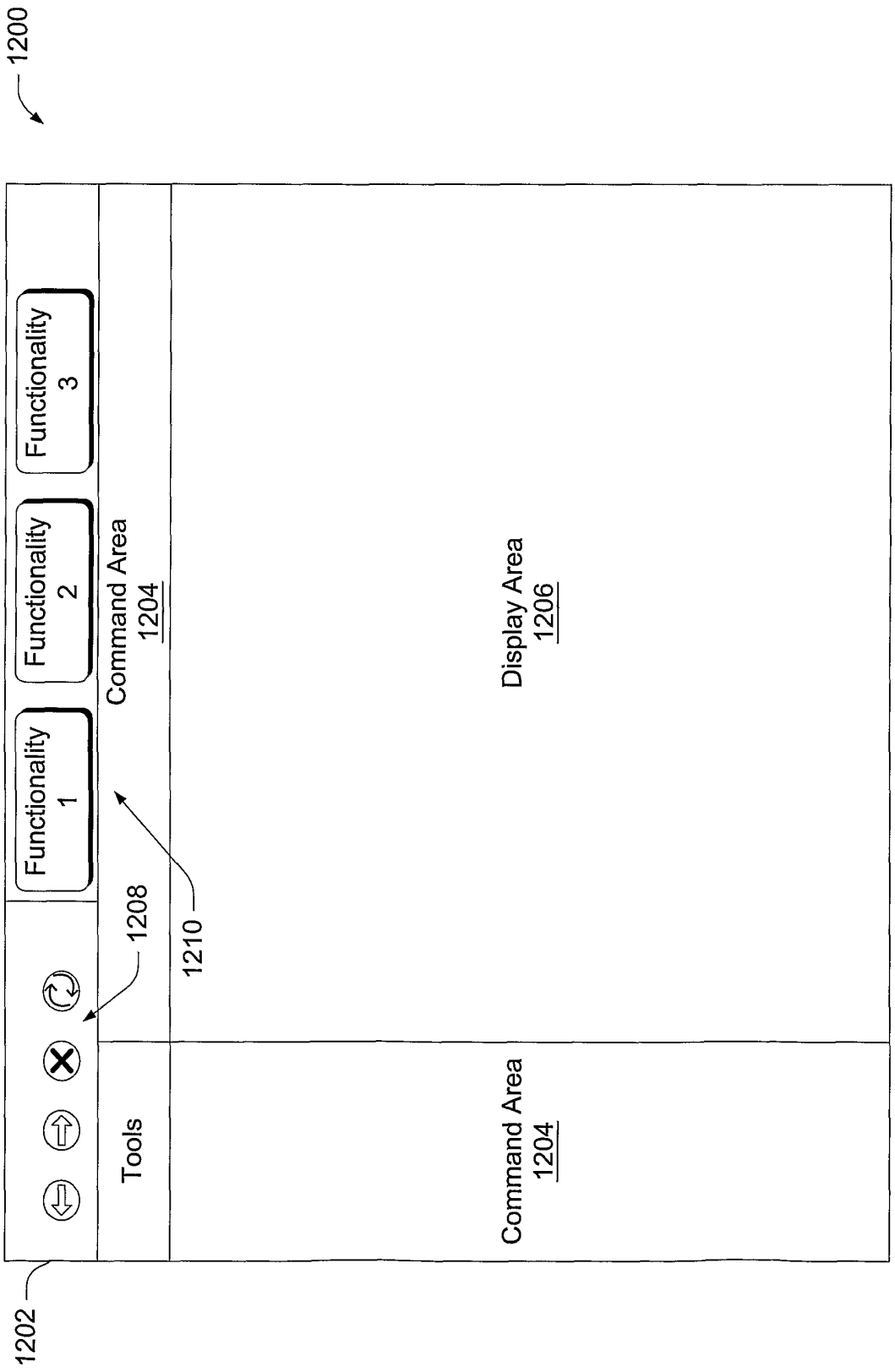

1200
Functionality 1
Functionality 2
Functionality 3
Command Area 1204
Tools
1208
1210
Display Area 1206
Command Area 1204
1202

Fig. 12

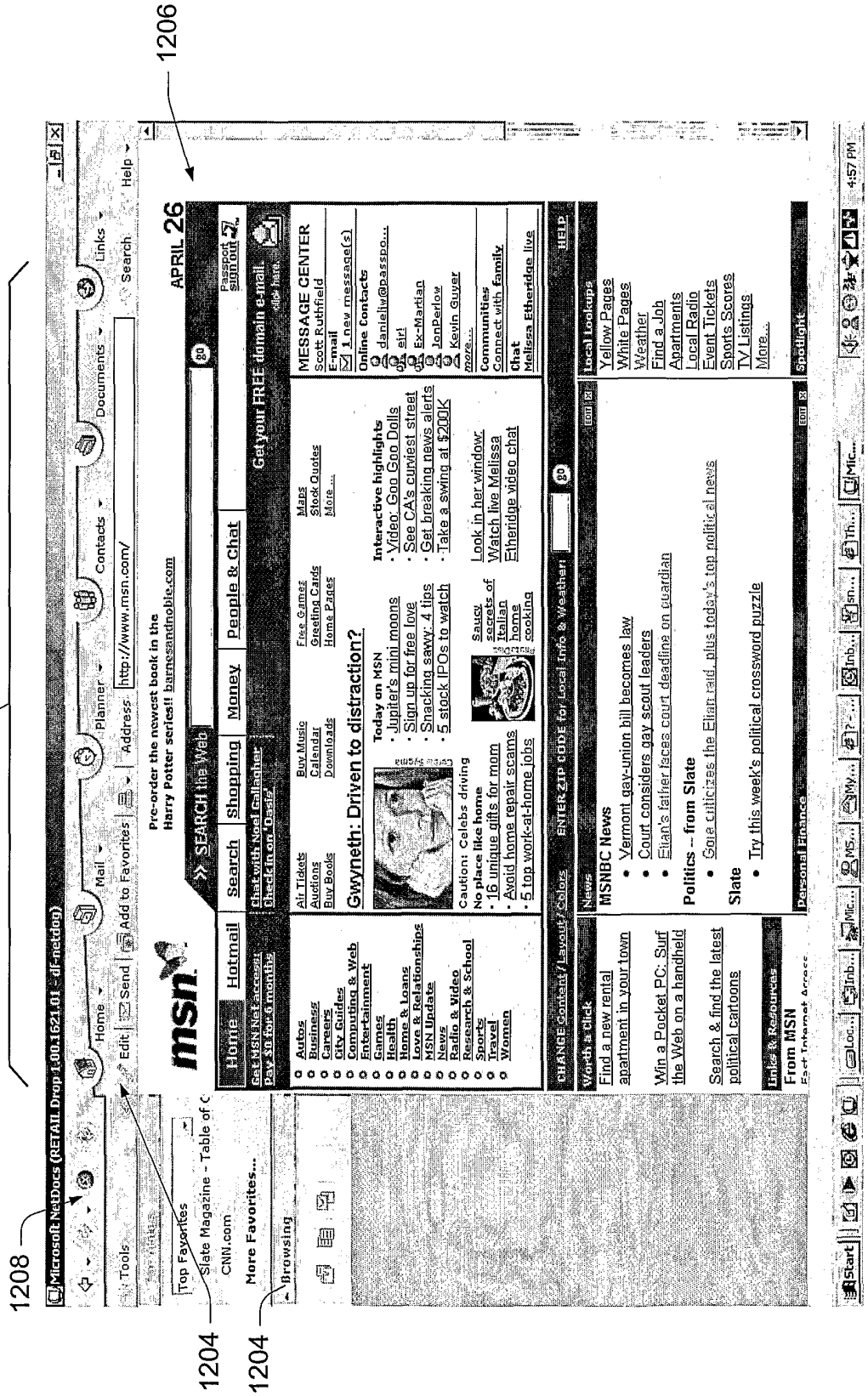

1206
1210
1208
1204
1204
msn
Home
Hotmail
Search
Shopping
Money
People & Chat
APRIL 26
Gwyneth: Driven to distraction?
MESSAGE CENTER
MSNBC News
Politics -- from Slate
Slate

Fig. 13

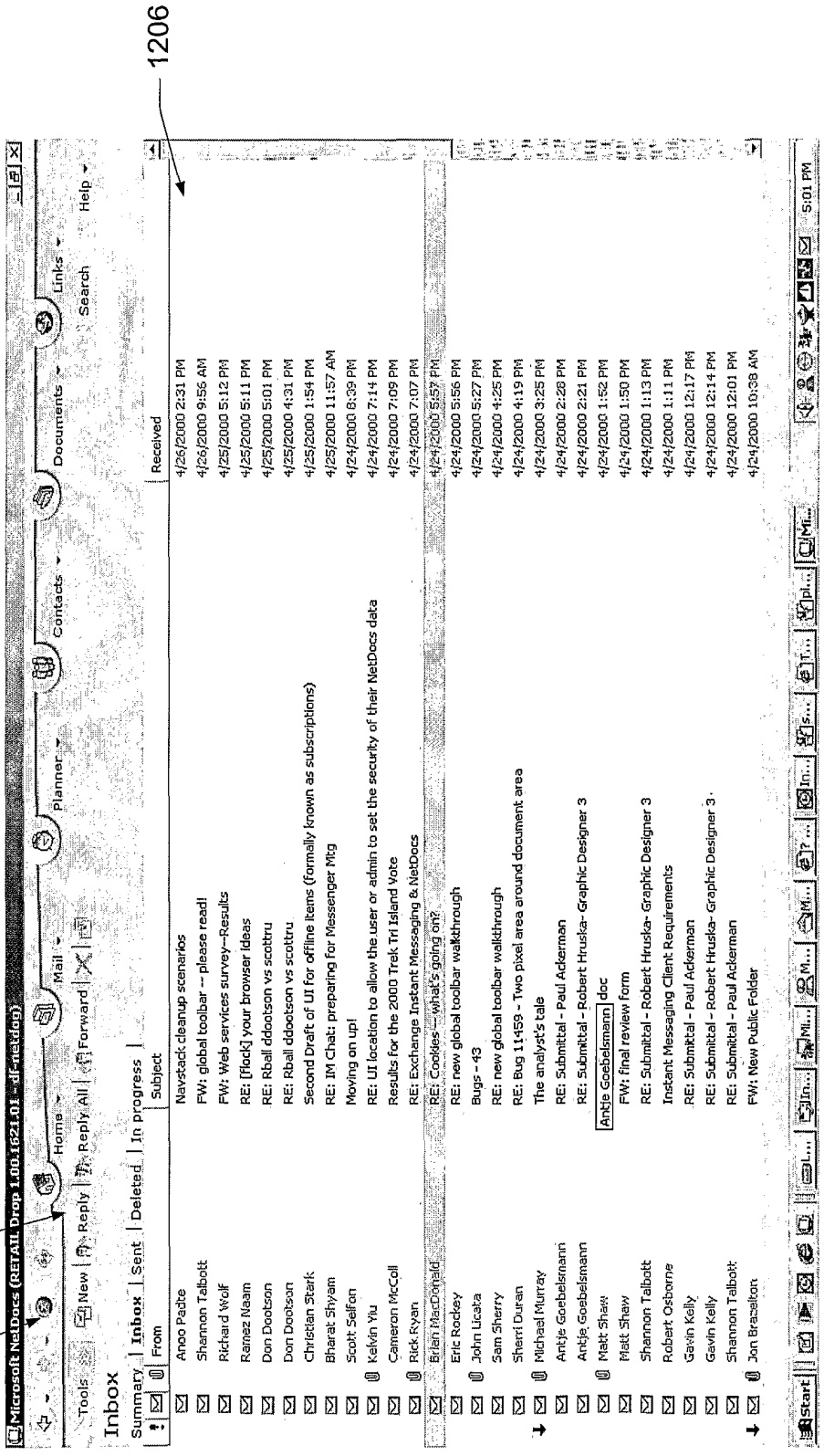

1210
1208
1204
1206
Microsoft NetDocs (RETAIL Drop 1.00.1621.01) - dFriedlog
Home
Mail
Planner
Contacts
Documents
Links
Tools
New
Reply
Reply All
Forward
Search
Help
Inbox
Summary
Inbox
Sent
Deleted
In progress
From
Subject
Received
Start
5:01 PM

Fig. 14

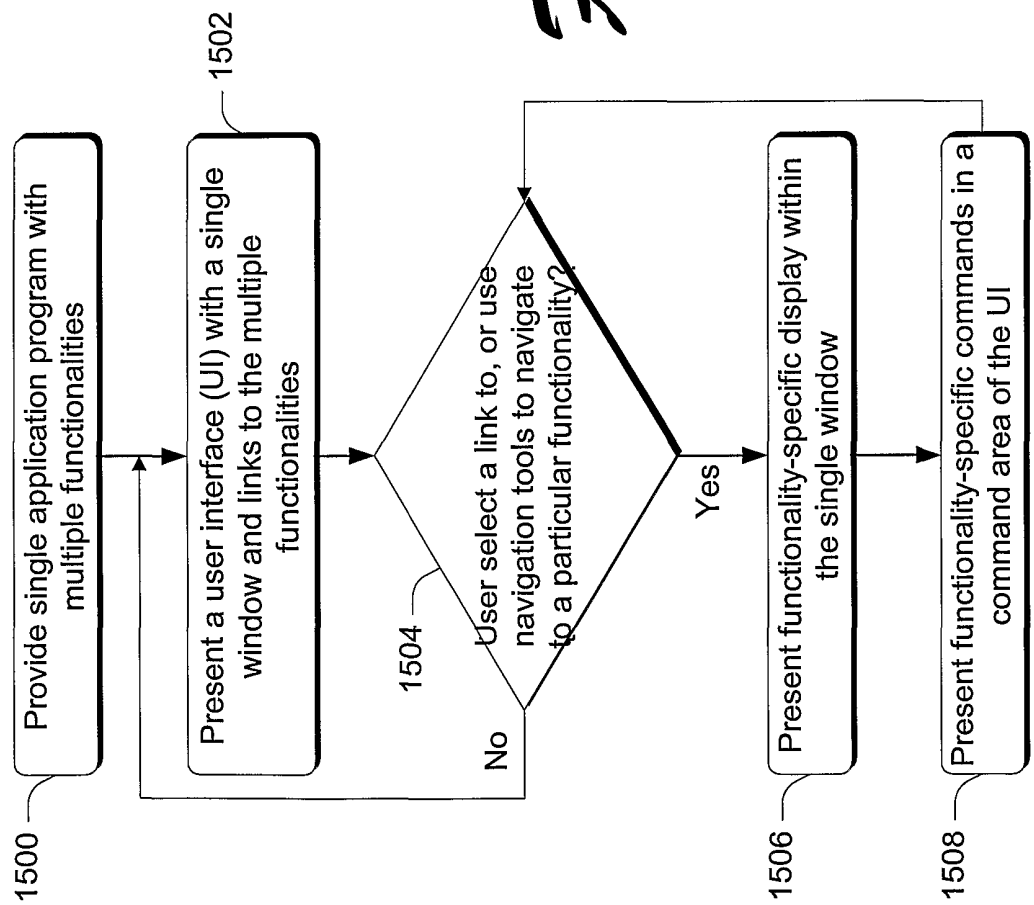

Provide single application program with multiple functionalities
1500
Present a user interface (UI) with a single window and links to the multiple functionalities
1502
User select a link to, or use navigation tools to navigate to a particular functionality?
1504
No
Yes
Present functionality-specific display within the single window
1506
Present functionality-specific commands in a command area of the UI
1508

Fig. 15