KATRIN – Keyboard Assisted TRaversal and Navigation

What problem does the invention solve?

Some modern web applications (including the VMware vSphere H5 Web Client) attempt to enhance their usability by providing keyboard shortcuts for commonly performed actions by the users.

The problem is that those shortcuts are provided when the user has selected certain items on the screen (i.e. list items, tree items, tabs). In order to use the shortcuts, the user needs to grab the mouse, use it to select the appropriate item on the screen, then leave the mouse and switch to the keyboard to invoke the shortcut combination.

This greatly diminishes the usefulness of keyboard shortcuts. Users usually don't perform that last part - once they grab the mouse, they select the item and then perform a right click with the mouse in order to invoke the "context" menu of the object. Thus keyboard shortcuts become more or less useless in great many scenarios.

We propose a solution that allows the user to conveniently and efficiently "navigate to" and "interact" with any element/item on the web page, using only the keyboard (no mouse, track-ball, touch-pad or touch-screen is involved). Once the user selects the corresponding element via the keyboard they can initiate the action directly by pressing the keyboard shortcut combination.

Briefly, summarize the invention and how it solves the stated problem

When a page is opened in the browser, a small JavaScript is loaded with it. This is the code that provides the functionality. The user can then start typing directly the text of the visual element they want to navigate to. A small rectangle is drawn around all the elements that match the typed “term”. The “currently focused” item also has a rectangle, but is drawn slightly differently than the rest of the matching items. The user (with the help of the arrow keys) can select the exact item they are looking for from all the “available” ones. Once the desired item is “currently focused”, the user can press Enter to force the behavior the element usually has on mouse click. This way a page can be navigated to without mouse availability. Prior to our invention, the user would be required to grab the mouse and click on the element. Methods are provided that the term search can also handle visual elements that don’t have text (i.e. are only icons). Methods are also provided to adequately include in the search only visual elements that the user is interested in, i.e. when there is a pop-up dialog or wizard the search for matching visual elements will be done only within the pop-up. Underlying visual elements will be excluded from the search.

You can check the supplied slides which illustrate the above idea.

Detailed Description

Prior know-how: The reader must be familiar what DOM is and what elements it has (https://www.w3.org/TR/DOM-Level-2-Core/introduction.html, http://www.w3schools.com/js/js_htmldom.asp)

- The web application is accompanied by a JavaScript code which provides all of the functionality. This means that the web page will include the file as part of its <script …></script> content.

- The search:
    - The user starts typing the text directly and the script triggers upon pressing the key. It then starts doing incremental search of the word just as you type.
    - Let us call what you type the “search term”.
    - It shows you what it currently searches for in a small rectangle in the upper right corner (the position should be configurable, i.e. right top, left top, left bottom, etc…)
    - It searches the browser’s DOM elements (only the visible ones) recursively
    - For every DOM element whose text contains the typed value
        - It visualizes all the found elements containing the currently typed text as “bounded” with a dashed yellow/orange border so that they are clearly visible.
    - It visualizes the “current/focused” element with a semi-transparent background color (besides the border) to make it stand out
    - The “current/focused” element is the one that the user will interact with if they press “Enter” immediately.
        - Upon search start, the first found element by the DOM traversal becomes the current element.
    - You can change the “current” element by pressing the “up/down/left/right” keys (the keyboard keys should be configurable). The closest “bounded” element will be filled-up and become the new “current” element.
    - If you press “Enter” at this point (*this should be a configurable key, e.g. some users may prefer “Space”*), the code will invoke the “click” event on the selected DOM element. This way, the element will assume the user has clicked with the mouse on it. This is what will make the user “interact” with the element, e.g.
        - If the element was, say, an item in a bigger control (i.e. list item, tree item) – mouse click simulation will make the element selected in the list which is what is expected from it – just as if the user actually clicked on it with the mouse.
        - If the element was a “tab” on the screen – mouse click will cause it to become focused and visualize its content which is what is expected from it – just as if the user actually clicked on it with the mouse
        - If the element was a “link-like” element that opens a new screen or page – triggering a click event will trigger its default behavior – just as if the user actually clicked on it with the mouse.
        - The overall idea is since web applications are 100% designed for mouse interaction – simulating the mouse click on the “current” element will make it behave exactly as expected by the users.
        - All the bounded rectangles are cleared and the search state is reset, allowing the user to continue their drill down on the newly opened interface parts afresh.
    - Pressing “Shift + Enter” (*this should be a configurable key, e.g. some users may prefer “Options Menu” key*), will invoke “click” (but with the right mouse button) event on the selected DOM element. This way the element will assume the user right-clicked with the mouse over it. If the application provides some context dependent menu – the user will quickly pop it up.
    - Pressing “Escape” key, clear the current search - removes the rectangle and clears all the bounded elements indication.

- Focusing on input fields
    - When “input” HTML elements are present in the page, it is expected that the user should be able to navigate to these the same way as to any other visual element. But “input” elements don’t have text. They look like empty rectangles waiting for the user to click on them and start entering text input. These are however almost always accompanied by a “nearby” text/label showing the purpose of the input field (some sample labels are: “username”, “address”, “phone” …).
    - Therefore the user can start typing those nearby strings, the search algorithm above will discover them and visualize them as “bounded”. But when the user makes one of them “focused” – additional scan is done to all children and sibling DOM elements of this “focused” text element. If any of the children or siblings turns out to be an “input” filed it will be visualized with the “bounding rectangle” too and be navigable via the arrow keys – so that the user can make it “current/focused”.
    - Once an “input” element is visualized as “current”, pressing “Enter” on it will invoke “focus” instead of “click” HTML event – therefore the element will gain the keyboard focus and the user can start typing whatever they like in this input field.
    - When the keyboard focus is in the input field, whatever the user types will not rigger any searches anymore but will end-up as a text in the input field. To go back to the search behavior, the user must somehow “leave” the input field. This is done with the help of the TAB key. In a standard HTML page, if TAB is pressed inside an input field, the browser moves the focus to the closest visual element in the HTML DOM tree that can accept focus. As a result, the user can now start typing the new search term and the navigation code will kick-in and do its usual job.
- Navigation via icons/images
  Sometimes web pages provide visual elements (icons, images) that don’t have text but when clicked, will contain some functionality on the click event. If those visual elements are equipped with tooltips and the content of the tooltip contains the search term, those elements will also be visualized with bounded rectangle and be part of the possible user interaction as described above (i.e. they can be focused via the arrow keys).
    - Usually tooltips are defined in the DOM’s elements “title” property – so we search for it here – but the attributes to search for should be configurable to allow for more fine-grained customization.
- Popups: Popups or dialogs are visual elements on the page that look like floating on top of the other. Once these are visualized, the natural user expectation is to interact only with items within these dialogs. Therefore, the “keyboard interaction” described above should be made to work only with the items in the “popup” and conveniently ignore all other items in the page.
    - To achieve the above, we require that the search algorithm is told/configured how pop-ups are structured in the DOM (i.e. a pop-up may be defined as a DIV element with particular CSS style and z-order)

- The search, when invoked, first scans the DOM for elements matching the pre-configured structure.
- If such DOM element is found the regular search for items is performed only within this sub-part of the DOM and the rest is left unsearched.
- Popups over other popups (over other popups, etc.) - to ensure that the search is done only in the top-most pop-up, the popup mechanism of the web application must put the DOM element signifying the top most pop-up before any other pop-up. This way the above “popup” search will enter the top-most DOM element marking a “pop-up” first and ensure consistent user experience searching in the “top most” pop-up visual hierarchy.

How have others tried to solve the problem and in what ways have their solutions been inadequate

The existing web browsers "search" functionality is somewhat similar to our approach. When you press Ctrl-F it lets you search text on the screen. In "search" mode it allows you to press "Enter" to see the next text that contains the searched text. Still, it only presents user with visual indication where the searched text is and does not provide any semantics on it (it does not become selected, you cannot simulate click, etc...). That means the existing “search” is practically useless for the purpose of navigation.