The Hotkey Palette: Flexible Contextual Retrieval of Chosen Documents and Windows

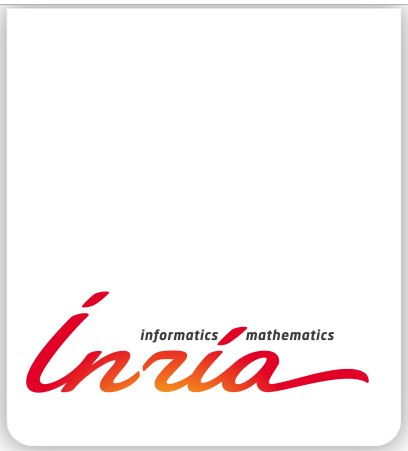
Jonathan Aceituno, Nicolas Roussel

The Hotkey Palette: Flexible Contextual Retrieval of Chosen Documents and Windows

Jonathan Aceituno, Nicolas Roussel

RESEARCH REPORT
N° 8313
June 2013
Project-Team Mint

ISSN 0249-6399 ISRN INRIA/RR--8313--FR+ENG



The Hotkey Palette: Flexible Contextual Retrieval of Chosen Documents and Windows

Jonathan Aceituno, Nicolas Roussel

Project-Team Mint

Research Report n° 8313 — June 2013 — 10 pages

Abstract: We present the Hotkey Palette, a quasi-modal interface enabling quick retrieval of chosen documents and windows by defining and triggering keyboard shortcuts either on the physical keyboard or with an on-screen keyboard. The Hotkey Palette improves on previous work by providing flexible contextualization of shortcuts that leverages document hierarchies, and by merging document and window retrieval in a single interface. The report describes the design and implementation of the interface and presents novel use cases for document and window management.

Key-words: on-screen keyboard, document retrieval, interface customization, window management

RESEARCH CENTRE
LILLE – NORD EUROPE

Parc scientifique de la Haute-Borne
40 avenue Halley - Bât A - Park Plaza
59650 Villeneuve d'Ascq

Hotkey Palette: Récupération flexible de documents et de fenêtres choisis

Résumé : Nous présentons Hotkey Palette, une interface quasi-modale permettant la récupération rapide de documents et fenêtres choisis à travers la définition et l'utilisation de raccourcis clavier, soit sur le clavier physique, ou bien sur un clavier virtuel affiché à l'écran. Hotkey Palette contribue à l'état de l'art en proposant une méthode flexible de contextualisation des raccourcis qui s'appuie sur les hiérarchies de documents, et en réunissant la récupération de documents et de fenêtres en une seule interface. Ce rapport décrit les principes de conception et l'implémentation de Hotkey Palette et présente des cas d'utilisation novateurs dans le cadre de la gestion de documents et de la gestion de fenêtres.

Mots-clés : clavier virtuel, récupération de documents, personnalisation d'interface, gestion de fenêtres

1 Introduction

Users of Personal Computers interact with a large number of resources[1] to do their work. To handle their different tasks, they need their documents to be readily available, and as the number of activities and documents increase, systems must offer adequate support for quick retrieval of these resources.

Document retrieval is primarily done by using a file browser, but navigation in a folder hierarchy takes a long time [3]. The desktop, the Windows Start Menu, OS X's Dock and similar *quick retrieval facilities* help decrease retrieval times by providing customizable areas on which shortcuts to chosen resources or the resources themselves can be placed. They are always available and thereby useful to keep things close at hand. But there is no way to indicate which resource is useful for which task, and the more things there are, the more time it takes to find the documents relevant to the task at hand in these places.

Quick access to specific resources can also be be provided by keyboard shortcuts. But the facilities provided to define these shortcuts are usually hard to use, and remembering them all is difficult. Graphical representations reifying the shortcuts and supporting direct manipulation can help. Touch-display keyboards (TDK [5]) would make it possible to create a keyboard shortcut by dragging a document to the desired key, for example. But such keyboards are currently uncommon and such a simple approach does not scale: one not only needs shortcuts but also ways to organise them.

The way people configure quick retrieval facilities depends on the context in which resources will be used [17]. Some resources pertain to specific projects while others are more general, and the users' workarounds to put resources close to where they might need them, as described in [17], suggest a lack of support for resource contextualization in current retrieval facilities. When all the needed documents for a task are found and open, the problem furthermore shifts from document retrieval to window management. Unlike document retrieval facilities, many window managers support the notion of task context by proposing window grouping [12] or virtual desktops [7]. However, window managers usually do not take the user's previous efforts on resource organization into consideration and require additional management that could be avoided.

Recent systems [2, 16, 8] help solve the contextualization problem by explicitly representing the activities used to organize resources and work around them. Activity-centric environments help effectively manage access to related windows *and* documents using the same notion of activity. For example, Giornata [16] not only maps windows of different activities to different virtual desktops, but also associates the files on each desktop to the relevant activity, which is particularly appreciated by users. Yet activities have blurry boundaries and users have trouble defining and separating them [1]. Moreover, activity-based approaches do not support finer-grained (linking a document to another) or coarser-grained (associating a document to several activities) contextualizations.

In this report, we present the Hotkey Palette (Figure 1), a quick retrieval facility that uses hotkeys and makes them visible and configurable through a quasi-modal [10] always-available on-screen keyboard. This facility contributes to the state of the art in three ways:

- it supports TDK-like interaction using standard keyboards;
- it provides persistent and integrated access to local windows and files and other online resources;

[1] In this report, the resources or documents we consider are local files, folders and applications, but also remote web pages, and, more generally, anything that can be identified with an uniform resource locator (URL) [4].

- it provides flexible control over contextualization by leveraging existing resource hierarchies.

The rest of this report is organized as follows. We first describe the design, the interaction techniques and the implementation of the Hotkey Palette. We then describe different usage patterns and finally conclude with a description of future work.

2 The Hotkey Palette

The Hotkey Palette (Figure 1) uses an on-screen keyboard to visualize and configure persistent shortcuts to access chosen documents and windows. It appears when the user holds a particular modifier key. In our implementation, we chose the *Fn* key of Apple keyboards, to avoid interfering with system shortcuts. The Hotkey Palette allows to access documents and windows by pressing associated keys while holding the modifier, and disappears when it is released. In this section, we explain how contextualization works, describe design of the Hotkey Palette and the interaction techniques it affords before detailing its current implementation.

2.1 Supporting contextualization

The following example describes our approach to the problem of resource contextualization that was described in the introduction, using the notions of *contextual shortcuts* and *hierarchical specialization* that we will detail later on.

Imagine a user, Alice, filing her projects on a local hard disk using the following hierarchy: the `Documents` folder contains two folders, `Work` and `Personal`, that each contain folders for her

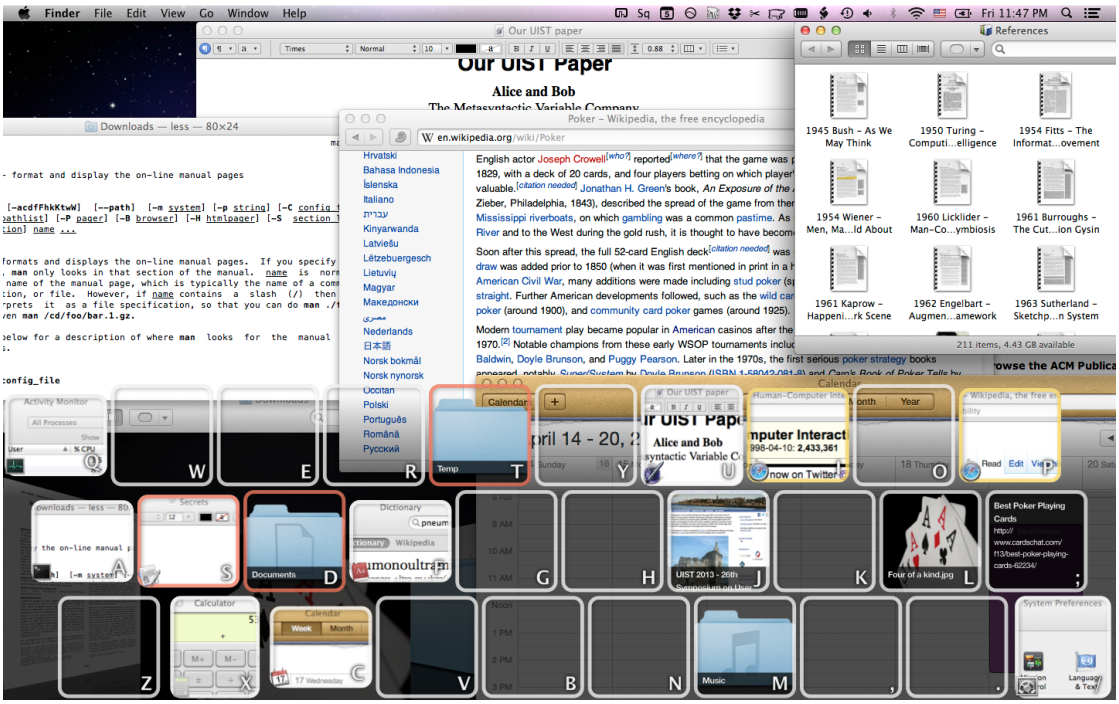


Figure 1: The on-screen keyboard of Hotkey Palette displayed on top of other windows.

work and personal projects respectively. She already has a *global* hotkey providing a shortcut to her `Documents` folder from anywhere using *Fn+D*. But she has two different to-do lists, for work and personal projects, and wants to have them handy. She creates two *contextual* shortcuts mapped on *Fn+T*: one in her work folder mapping it to her work to-do list, and the other in her personal folder mapping it to the personal one. Thanks to *hierarchical specialization*, pressing *Fn+T* while working on a work project (i.e. any document in `Work` or its sub-folders) will thus open the work to-do list, but pressing the same *Fn+T* while working on a personal project (i.e. i.e. any document in `Personal` or its sub-folders) will open the personal to-do list.

One of Alice's personal projects is to maintain the web site of the local poker league, and she frequently receives e-mails from members prompting her to announce tournaments on this site. To support her workflow, she has created a contextual shortcut to the local copy of the web site's announcements page associated to her mail application, and another to a synchronization script associated to the entire web site's local copy (i.e. the enclosing folder). When she receives an announcement mail, she uses the former shortcut to quickly access the announcements page and modify it. From this document, through *hierarchical specialization*, she can use the latter to synchronize the online web site with the modified local copy.

We can see in this example that contextual shortcuts differ from global ones in that they are associated to a specific resource (interaction details are discussed later). They are only available when that resource is the current object of interest (the one displayed in the active window), thus providing context-specific access to others. *Hierarchical specialization* makes it possible to support the same contextual shortcuts on different related resources by exploiting their hierarchical relation.

In the above example, several shortcuts are mapped to the same key. To determine which shortcut is the most relevant to the current resource of interest, we use the following relevance order:

- contextual shortcuts pertaining to this object (X),
- contextual shortcuts pertaining to any resource Y such that Y is an ancestor of X[2] (the closer, the better),
- global shortcuts.

Hierarchical specialization is the outcome of the above rule: when a contextual shortcut is associated to a folder, it will be available by default for all the documents it contains.

2.2 Interaction

In the example above, the user created and triggered shortcuts. Here, we detail the interaction techniques that support these functionalities and more advanced ones.

2.2.1 Configuring a hotkey

Pressing *Fn* and some unused key X creates a global shortcut mapping *Fn+X* to the current resource of interest. Holding *Fn+X* while clicking on an on-screen resource creates a global shortcut to it. Holding *Fn+Shift+X* when clicking instead results in a contextual shortcut to it associated to the current resource of interest. Both elements are highlighted in different colors while the keys are held (the current resource of interest and the one under the mouse pointer).

[2]Every document has a URL which path can be used to determine the relative location to another, if they live in the same hierarchy. The folder `file:///Documents` is an ancestor of the file `file:///Documents/Work/UIST Paper.doc` for example.

2.2.2 Triggering a hotkey

To trigger an existing shortcut mapped on a particular key, e.g. *X*, the user must hold the *Fn* modifier and press *X*. This brings the associated resource to the foreground. As an alternative, the user can click on the desired key while the Hotkey Palette is shown. Since shortcuts are persistent, the triggered one may be associated to a resource that is not available on-screen anymore. In this case, the resource is re-opened, if possible. If not, a system beep is issued.

2.2.3 Visual design

The Hotkey Palette appears on the bottom of the user's main display as a semi-transparent on-screen keyboard sized to fill the whole horizontal space (Figure 1). Only the keys on the three letter rows[3] are used. This way, each object associated to a key is displayed in a square taking up about 1/11 of the horizontal resolution of the main display (174 pixels on a 1920x1080 display, 93 pixels on a 1024x768 display).

What is displayed on a key depends on the resource associated to it. If there is no shortcut, the key is just displayed with its character as an empty square with a white border (Figure 3a). If the key is associated to a local or remote document (e.g. a web page), an up-to-date preview is displayed together with its title, filling the whole square (Figure 3b). If a preview is not available, its title and/or URL is displayed instead (Figure 3c). Finally, if the key is associated to a window, its aspect ratio is modified to fit the window's, and an up-to-date and unscaled preview of the window centered on its title is displayed together with the application icon (Figure 3d). We chose this presentation because we found it easier to distinguish between similar windows than with scaled-down screenshots (Figure 2).

A key associated to a window that does not exist anymore is yellowed if it can be re-opened (Figure 3e). However, a key is grayed if it is associated to a window or resource that cannot be found, because it has been deleted for example (Figure 3f).

The border of a key indicates if the shortcut is global (white border) or contextual. A yellow border (Figure 3g) indicates that the shortcut is associated to the currently active resource, while a red border (Figure 3h) indicates that it is associated to one of its hierarchical ancestors.

2.2.4 Graphical interaction

When the Hotkey Palette is visible, advanced graphical interaction with the shortcuts is possible. They can be moved around the keyboard using drag-and-drop. Users can drag a document from any application, hold *Fn* and drop it on a key to create a global shortcut (as before, holding *Fn+Shift* results in a contextual shortcut related to the current resource of interest). Dropping a document on a key containing an application or a script will launch the script or application using the dropped document as a parameter. Dragging a shortcut to a document from a key and dropping it in an external window will have the same effect as dropping the document itself. A right click on a key shows a menu with options to delete the shortcut and, for contextual ones, to bring up the resource defining the context.

2.3 Implementation

Our prototype is implemented in Objective-C with Cocoa and runs under Mac OS X. We used the Quartz Window Services API to retrieve window information, and the Accessibility API to inspect objects under the mouse, as well as to get specific information on the contents of a window, such as the document it represents. We also used the bookmarks API to make shortcuts resilient

[3] In a standard QWERTY keyboard, these rows correspond to characters `Q` to `P`, `A` to `;`, and `Z` to `/`.

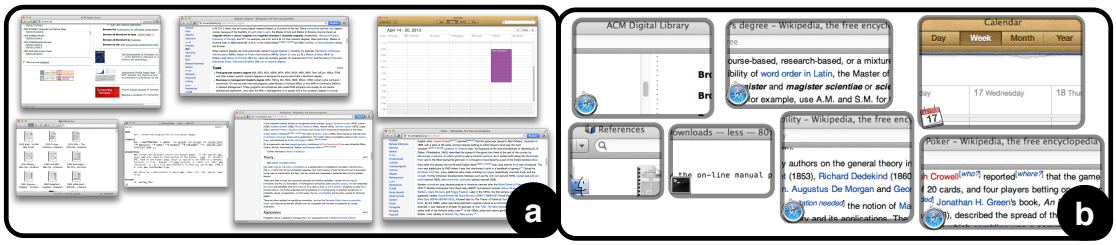


Figure 2: Two different window visualizations: (a) Exposé-like, scaled down; (b) Unscaled, top-centered.

to renamed or moved documents, ensuring that they can always be opened, and the previews of local documents and remote web pages are generated using the Quick Look API and WebKit, respectively.

To ensure persistence over time, for example to be able to re-open a document that has long been closed, the program saves specific cues about the corresponding resource and its context whenever a shortcut is created. The exact nature of these cues depends on the underlying accessibility API. When the program loses track of a window, it tries to match the saved cues to each of the opened ones to find it again. If it cannot find it, it tries to locate the resource that was displayed in the window and to re-open it.

3 Usage patterns

In this section, we present some of the usage patterns that the distinctive features of the Hotkey Palette enable, namely: reminding the user of something to do, monitoring changes, improving web page revisitation, and cycling between specific documents or windows.

3.1 Reminders and monitors

Shortcuts to resources can be used as reminders of related actions to execute in the future. These reminders can be contextual shortcuts, so that the user chooses *when* to be reminded of them. For example, Alice may create a contextual shortcut associating a photograph of her coworker Bob to her `Work` folder so that the next time she works on a project in this folder and brings up the Hotkey Palette, she will incidentally be reminded of inviting him to the next poker tournament.

Because it shows large and up-to-date previews, the Hotkey Palette can also be used to monitor changes on documents and web pages. Although the current implementation does not indicate whether the document has changed since the last visit, users can still benefit from the up-to-date preview to check, for example, if Alice has indeed updated the poker league's website as expected.

3.2 Alternative bookmarks

Web page revisitation is primarily supported in web browsers by bookmarks and history. The Hotkey Palette can also act as an alternative bookmark facility, or an extended version of PageLinker [15]: here, one can not only associate a web page to another, but contextual links can also be created between web pages and any local resource, in both directions. Like with PageLinker, the query part of the URL is not taken into account for context to avoid tying

Figure 3: Closeup on the top left keys of the Hotkey Palette above the active window: (a) empty key; (b) document preview; (c) document title when the preview is not available; (d) window preview; (e) yellowed preview of a closed window; (f) grayed preview of a deleted document; (g) contextual shortcut associated to the active window (yellow border); (h) contextual shortcut associated to an ancestor of the document shown in the active window.

shortcuts to a particular web session. Hierarchical specialization works in this case because web URLs have paths.

3.3 Local cycling

In his description of the *memex* [6], Vannevar Bush defined associative trails as a chain of documents linked together, sometimes branching off to side trails. The imagined interactions with these trails involved only hitting a few keys. While the Hotkey Palette is not designed to handle trail navigation, this principle can still be useful to quickly travel along a trail of related resources by creating contextual shortcuts mapped to the same key and associating these resources.

Alice could use this approach to create a personnal webring linking her preferred poker news sites, for example. But she could also use it to cycle between a specific list of applications. She could use the *C* key to cycle between all her communication applications, for example. To do so, she would first create a global shortcut mapping that key to the contact list of her instant messaging application. She would then define a contextual shortcut from this window to her mail application, using the same key. She would continue with the websites or applications of her various social networks. Pressing *Fn+C* on any of these resources would then bring up the next one, the global shortcut to the instant messaging application closing the ring.

4 Conclusion and future work

In this report, we presented the Hotkey Palette, a quick retrieval facility based on user-defined shortcuts with the following distinctive features: quasi-modal interaction with an on-screen keyboard, persistent and integrated access to local windows and files and other online resources, and flexible contextualization based on resource hierarchies. We detailed the peculiarities of our design, the associated interaction techniques and our current implementation. Finally, we showed how the technique allows novel uses in window and document management.

By merging two input modalities, pointing and typing, on a shared representation, on-screen keyboards may provide benefits for intermodal learning [14], as syntactically similar actions (e.g. pressing a key vs. clicking an on-screen key) have the same effects. In addition, they provide a spatially stable layout, allowing users to rely on their special memory, which could lead to faster retrieval times [11, 13]. Both of these hypotheses require further investigation. In the future, we would also like to investigate how the design of the Hotkey Palette could be used to improve learning of application hotkeys, and how it compares to recent work [9].

References

[1] J. E. Bardram. *Beyond the Desktop Metaphor: Designing Integrated Digital Work Environments*, chapter From Desktop Task Management to Ubiquitous Activity-Based Computing, 224–259. MIT Press, 2007.

[2] J. E. Bardram, J. Bunde-Pedersen, and M. Soegaard. Support for activity-based computing in a personal computing operating system. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, 211–220. ACM, 2006.

[3] O. Bergman, S. Whittaker, M. Sanderson, R. Nachmias, and A. Ramamoorthy. How do we find personal files?: The effect of os, presentation and depth on file navigation. *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, 2977–2980. ACM, 2012.

[4] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifier (uri): Generic syntax. RFC 3986, Jan. 2005.

[5] F. Block, H. Gellersen, and N. Villar. Touch-display keyboards: Transforming keyboards into interactive surfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, 1145–1154. ACM, 2010.

[6] V. Bush. As we may think. *Atlantic Monthly*, 176:101–108, 1945.

[7] D. A. Henderson, Jr. and S. Card. Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Trans. Graph.*, 5(3):211–243, July 1986.

[8] S. Houben, J. Vermeulen, K. Luyten, and K. Coninx. Co-activity manager: Integrating activity-based collaboration into the desktop interface. *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '12, 398–401. ACM, 2012.

[9] S. Malacria, G. Bailly, J. Harrison, A. Cockburn, and C. Gutwin. Promoting hotkey use through rehearsal with exposehk. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2013.

[10] J. Raskin. *The Humane Interface: New Directions for Designing Interactive Systems.* ACM Press/Addison-Wesley Publishing Co., 2000.

[11] G. Robertson, M. Czerwinski, K. Larson, D. C. Robbins, D. Thiel, and M. van Dantzich. Data mountain: Using spatial memory for document management. *Proceedings of the 11th annual ACM symposium on User interface software and technology*, UIST '98, 153–162. ACM, 1998.

[12] G. Robertson, E. Horvitz, M. Czerwinski, P. Baudisch, D. R. Hutchings, B. Meyers, D. Robbins, and G. Smith. Scalable fabric: Flexible task management. *Proceedings of the working conference on Advanced visual interfaces*, AVI '04, 85–89. ACM, 2004.

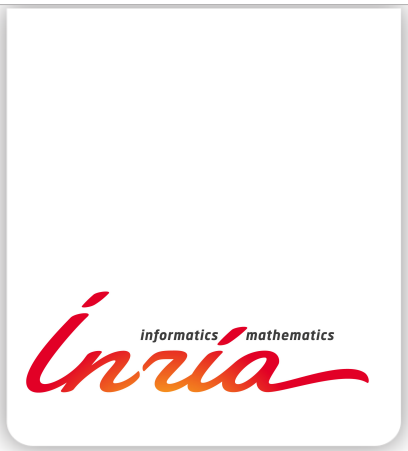
[13] J. Scarr, A. Cockburn, C. Gutwin, and A. Bunt. Improving command selection with commandmaps. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, 257–266. ACM, 2012.

[14] J. Scarr, A. Cockburn, C. Gutwin, and P. Quinn. Dips and ceilings: Understanding and supporting transitions to expertise in user interfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, 2741–2750. ACM, 2011.

[15] A. Tabard, W. Mackay, N. Roussel, and C. Letondal. Pagelinker: Integrating contextual bookmarks within a browser. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, 337–346. ACM, 2007.

[16] S. Voida, E. D. Mynatt, and W. K. Edwards. Re-framing the desktop interface around the activities of knowledge work. *Proceedings of the 21st annual ACM symposium on User interface software and technology*, UIST '08, 211–220. ACM, 2008.

[17] A. Zacchi and F. Shipman. Personal environment management. L. Kovács, N. Fuhr, and C. Meghini, editors, *Research and Advanced Technology for Digital Libraries*, volume 4675 of *Lecture Notes in Computer Science*, 345–356. Springer Berlin Heidelberg, 2007.



RESEARCH CENTRE
LILLE – NORD EUROPE

Parc scientifique de la Haute-Borne
40 avenue Halley - Bât A - Park Plaza
59650 Villeneuve d'Ascq

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399