All you need is log

Nicolas Roussel
Projet In Situ
LRI & INRIA Futurs
Bât 490, Université Paris-Sud
91405 Orsay Cedex, France
roussel@lri.fr

Aurélien Tabard
Projet In Situ
LRI & INRIA Futurs
Bât 490, Université Paris-Sud
91405 Orsay Cedex, France
tabard@lri.fr

Catherine Letondal
Pôle Informatique
Institut Pasteur
28, rue du Docteur Roux
75724 Paris Cedex 15, France
letondal@pasteur.fr

ABSTRACT

Experiences in our Micromégas project have shown that a detailed interaction log can be valuable to both users and designers of interactive systems. In this paper, we describe two different attempts at logging traces of Web activity, the first by instrumenting a specific browser, Camino, and the other by creating a cross-platform Firefox extension. We briefly discuss some of the issues brought by these developments and suggest directions for future work.

Categories and Subject Descriptors

H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*Web-based interaction*; K.6.2 [**Management of Computing and Information Systems**]: Installation Management—*Performance and usage measurement*

General Terms

Measurement, Design, Human Factors, Standardization

Keywords

Web use, interaction log

1. LOG, LOG, LOG (INTRODUCTION)

The quantity and diversity of digital information that we produce, receive and download on a daily basis is becoming increasingly difficult to manage. The Micromégas[1]project aims to develop methodologies and tools to better understand how users manage their documents and applications and to design new metaphors and interaction techniques to facilitate these tasks. Recent advances in this project motivated closer examination of the problem of automatic capture of qualitative and quantitative information about a users' computing activity. As others have done in the past (e.g. [6]), we have implemented a series of applications to monitor and visualize window management, file management and network-related activity [8, 3].

Initially, we implemented the monitoring applications to generate detailed descriptions that would aid in understanding user activities. Yet, it quickly became clear that the collected information would also be useful to the users themselves, and to the software design community. In particular, a detailed log of his interactions with a system could help a user review previous work, return to a previous state and re-execute complex commands or send them to another person [9]. Providing computing systems with long term interaction memory could also make it easier for users to retrieve a particular data or context by taking advantage of their episodic memory [7].

The particular group of users we are working with is constituted of biologists from Pasteur Institute who use the Web on a daily basis for information retrieval and analysis. Typical tasks include retrieval of gene and protein sequences from Web databases and analysis using Web application portals. We identified a common difficulty: the pages providing data infrequently contain links to the tools required for their analysis, and traditional history and bookmark mechanisms provide insufficient support for retrieval of previously used tools or data sets. The problem in the biologist example illustrates a class of problems relating to Web navigation histories. As V. Bush states, the human mind operates by association, connecting items into a *web of trails* [1], but even modern navigation tools such as Web browsers take little advantage of these trails.

The notion of *detailed interaction log* is poorly supported by most existing applications. Lists of "recent items" are often available, but they usually present only a few items with little temporal context. Although some applications allow undo or redo for recent operations on a particular object (e.g. a document), very few remember these operations between sessions (e.g. after the document has been closed and re-opened). Web browsers provide both short-term and long-term history mechanisms through the *back* and *forward* buttons, personalized lists (*bookmarks* or *favorites*) and a *global history* list. Yet, while revisitation is the dominant Web activity and the *back* button is one of the mostly used interface components, global history lists remain rarely used [2, 11, 4].

The information that browsers store in their history file is quite limited (Figure 1). In particular, knowing only the last time (and possibly the first) a URI was visited makes it impossible to find other URIs that could have been visited at about the same time, earlier: if you frequently visit the same Web page, knowing you reached a document you're looking for from that page a month ago won't help you find it. The so-called *global history* is in fact only a subset of what we understand as the interaction log.

Our experiences in the Micromégas project have shown that a detailed interaction log can be valuable to both users and designers of interactive systems. Logging real users in

[1]`http://insitu.lri.fr/micromegas/`

	URI	title	VisitCount	LastVisitDate	FirstVisitDate	Referrer
Internet Explorer	x	x	x	x		
Mozilla and derivatives	x	x	x	x	x	x
Opera	x	x	x	x		
Safari	x	x	x	x		

Figure 1: Data stored in the global history file of popular browsers

their particular work context, however, proved harder than we anticipated. In the subsequent sections, we describe two different attempts at logging traces of Web activity, the first by instrumenting a specific browser, Camino, and the other by creating a cross-platform Firefox extension. We will then briefly discuss some of the issues brought by these developments and suggest directions for future work.

2. THERE'S NOTHING YOU CAN DO THAT CAN'T BE DONE

Camino is a Web browser that embeds the Mozilla layout engine[2] in a native Mac OS X interface written in Cocoa[3]. Our first attempt at logging a user's Web activity consisted in adding Objective-C code to selected functions of this browser. There was no reason for choosing this one against other browser, except that one of us was a little familiar with its compilation process. Our modifications use Cocoa's inter-process communication mechanisms to broadcast notifications to other interested applications. About sixty lines of code added to four functions (three files) were enough to generate seven types of notifications:

`setActive` **and** `setInactive` a document view, tab or window, acquires or looses the focus (e.g. when switching between tabs or windows)

`loadingCompleted` **and** `loadingFailed` a view succeeded or failed to load a new document (e.g. after activating a link, typing a URI or opening a file)

`downloadCompleted` **and** `downloadFailed` a remote file was successfully or unsuccessfully saved to the local disk (e.g. when activating a "Save as" or "Download link" command)

`closed` a document view is being closed

Applications that want to be notified simply need to subscribe to `fr.lri.insitu.micromegas.camino`, which can be done using only twenty lines of Python code, for example. Figure 2 gives an example of the kind of information attached to a `loadingCompleted` notification. Note that in addition to a timestamp, the URI and the title of the document that was just loaded, the application gets information about the user, the process and the view. It also has information about the function in the Camino source code that generated the notification.

Instrumenting Camino was both simple and elegant. Patching its publicly-available code and keeping our version synchronized with the CVS one was quite easy. Finding observation points inside the code was also relatively easy because of the Model-View-Controller design pattern followed

```
fr.lri.insitu.micromegas.camino loadingCompleted
    obsPoint: BrowserWrapper.mm/onLoadingCompleted
    process: 670
    timestamp.sec: 1139571047
    timestamp.usec: 139838
    title: LRI, projet In Situ
    uri: http://insitu.lri.fr/
    user.id: 501
    user.login: roussel
    user.name: Nicolas Roussel
    view: 120828592
```

Figure 2: Sample notification received by a Camino observer

by Cocoa. Keeping the added code as small as possible (i.e. only use it to generate notifications) and having the actual logging implemented in external applications allowed to compare various ways to record data and later make use of it. It makes it possible to implement the logging code in any language using any toolkit we want. A more complex notification service like D-BUS[4] would even make it possible to handle the notifications on a different machine. In a way, as the song says, using this approach *there's nothing you can do that can't be done.*

However, this approach has also disadvantages. One has to compile a specific version of Camino and distribute it. Although about half of our target users have Apple Computers, none of them are actually using this particular browser (they use Safari, Firefox or Internet Explorer). This was clearly a problem, since we wanted to observe users in their "natural ecology". Even if they had used Camino, replacing the browser executable can sometimes be troublesome, as incompatibilities arise between versions.

3. NOTHING YOU CAN SAY BUT YOU CAN LEARN HOW TO PLAY THE GAME

Our first experience with Camino gave us hope that a similar approach could be applied to another browser. We chose Firefox, the latest cross-platform and extensible browser from the Mozilla family. About half of the biologists we met were using that browser on OS X, Windows or Linux. Firefox has a built-in extension mechanism[5] that allows to easily create cross-platform add-ons that bring new functionalities to the browser and can be downloaded and installed by users in a few clicks. This time, we thus decided to create an extension rather than modify the browser's source code.

Firefox's user interface is written in XUL and JavaScript. XUL is an XML-based user interface description language, JavaScript is a cross-platform object-oriented scripting lan-

[2]`http://www.mozilla.org/newlayout/`
[3]`http://developer.apple.com/cocoa/`
[4]`http://www.freedesktop.org/Software/dbus`
[5]`http://developer.mozilla.org/en/docs/Extensions`

guage. Building a Firefox extension consists in creating XUL "overlays" to add or modify interface elements and attaching JavaScript handlers to respond to specific events on these elements. The extension programming interface also provides RDF support for storing and retrieving data.

We are currently developing navtracer[6], a Firefox extension designed to log the interactions between a user and his browser. This extension registers various event handlers to detect the opening or closing of tabs and windows, and the acquiring or loss of focus. It also tracks document changes and the relations between them. Event handlers append log data to a plain text file stored in the user's profile folder. Timestamps are systematically added to every record. One short-term goal for this extension is to provide enough data to visualize the paths followed by a user. Another goal is to evaluate the potential benefit of another extension we developed that allows biologists to create contextual links between web pages [10].

One problem we're facing is that extensions can only react to a predefined set of events occurring on interface components. To our knowledge, there is no way to request some code to be executed before or after a particular function of the browser is called. As a consequence, to trace function calls that have no associated XUL event, one needs to register handlers for events that might precede or follow these calls and use some clever heuristics. Failures are particularly hard to detect. Windows, for example, have an `onload` event handler, but no `onloadfailed`. To detect that the user failed to open a document, one has to register low-level handlers with all the components that might call the loading function (e.g. the *Go* and *Bookmarks* menus; the history or bookmarks sidebars; the bookmarks, location and search toolbars) and, when one of these handlers triggers, check if the window receives a `load` event.

In the end, although the navtracer extension is indeed cross-platform and easily installed, developing it is a time-consuming task. And learning how to play with JavaScript and XUL can be a very frustrating experience...

4. LOG IS ALL YOU NEED (CONCLUSION)

Accounts of a system's activity are much more valuable if generated from within the system rather than imposed or inferred from outside [5]. Finding the appropriate observation point and level of detail for describing the activity is a hard problem. Instrumenting a platform-specific browser like Camino by modifying its source code lets us precisely specify these two parameters. But doing so in a cross-platform way to better integrate with existing work practices is much harder. The Firefox programming interface was designed by browser developers who were not especially concerned with user activity logging. The price to pay for cross-platform compatibility is a generic framework that does not necessarily have the right abstractions.

Firefox developers are currently working on a project to improve access to history and bookmarks[7]. While their current proposal is pretty conservative, we believe that HCI researchers interested in logging Web activity should seize this opportunity to sensitize the Mozilla community and the W3C to their problems and goals. In particular, in addition to the existing scripting facilities, it seems that observing user activity would be much easier if the browsers also provided some generic notification facilities like the ones we added to Camino. This would also make it easier to write external applications that somehow need to cooperate with a browser[8].

Although we are mostly interested in using the interaction logs to enhance the navigation services provided by Web browsers, we can think of many other reasons for logging a person's Web activity. Designers of Web services or even HTTP servers might learn a lot about their own software by examining detailed browser interaction logs. The continuous recording of these logs poses a number of technical questions. Where and how are they stored? How long are they kept? Can they be accessed and processed in real-time? The analysis of the collected data might also pose some interesting Data Mining questions. However, our experience tells us that interesting ideas about how this data might be used won't necessarily come out of the logs themselves. If you're looking for inspiration, you don't need logs. All you need is a group of users.

[6]`http://navtracer.mozdev.org/`

[7]`http://wiki.mozilla.org/Places`

5. ACKNOWLEDGMENTS

The Micromégas project is funded by the French *ACI Masses de données*. The authors thank John L. and Paul M. for suggesting the titles used in this paper.

6. REFERENCES

[1] V. Bush. As we may think. *Atlantic Monthly*, 176(1):101–108, June 1945.

[2] L. D. Catledge and J. E. Pitkow. Characterizing browsing strategies in the world-wide web. In *Proceedings of the Third International World-Wide Web conference on Technology, tools and applications*, pages 1065–1073, New York, NY, USA, 1995. Elsevier North-Holland, Inc.

[3] O. Chapuis. Gestion des fenêtres : enregistrement et visualisation de l'interaction. In *Proceedings of IHM 2005, 17ème conférence francophone sur l'Interaction Homme-Machine*, pages 255–258. ACM Press, International Conference Proceedings Series, Septembre 2005.

[4] A. Cockburn, S. Greenberg, S. Jones, B. McKenzie, and M. Moyle. Improving web page revisitation: Analysis design and evaluation. *IT & Society*, 1(3):159–183, 2003.

[5] P. Dourish. Accounting for system behavior: representation, reflection, and resourceful action. In *Computers and design in context*, pages 145–170, Cambridge, MA, USA, 1997. MIT Press.

[6] A. Dragunov, T. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. Herlocker. Tasktracer: a desktop environment to support multi-tasking knowledge workers. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 75–82. ACM Press, 2005.

[7] M. Ringel, E. Cutrell, S. Dumais, and E. Horvitz. Milestones in Time: The Value of Landmarks in

[8]Skype, for example, has a simple API that allows external applications to take control of certain of its functions and receive high-level notifications (`http://share.skype.com/sites/devzone/`)

Retrieving Information from Personal Stores. In *Proceedings of Interact 2003, the 9th IFIP TC13 International Conference on HCI*, pages 184–191. IOS Press, Amsterdam, Sept. 2003.

[8] N. Roussel, J. Fekete, and M. Langet. Vers l'utilisation de la mémoire épisodique pour la gestion de données familières. In *Proceedings of IHM 2005, 17ème conférence francophone sur l'Interaction Homme-Machine*, pages 247–250. ACM Press, International Conference Proceedings Series, Septembre 2005.

[9] R. Salter, B. Shneiderman, B. Bederson, G. Rubloff, C. Plaisant, and A. Rose, editors. *History Keeping in Computer Applications: a Workshop*, Dec. 1999. http://www.cs.umd.edu/hcil/about/events/history-workshop/.

[10] A. Tabard. Conception d'un navigateur web spécifique pour la bio-informatique. Rapport de stage de Master, Université Pierre et Marie Curie, Paris 6, Septembre 2005. 67 pages.

[11] L. Tauscher and S. Greenberg. How people revisit web pages: empirical findings and implications for the design of history systems. *International Journal of Human Computer Studies*, 47(1):97–137, 1997.