All you need is log
Aurélien Tabard, Nicolas Roussel, Catherine Letondal

To cite this version:

HAL Id: hal-01299649
https://hal.archives-ouvertes.fr/hal-01299649
Submitted on 8 Apr 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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’ 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/
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 develop-
ments 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 lay-
out engine\(^2\) in a native Mac OS X interface written in Co-
coa\(^3\). 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 fa-
miliar with its compilation process. Our modiﬁcations use
Cocoa’s inter-process communication mechanisms to broad-
cast 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 win-
dow, acquires or loses 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 sub-
scribe 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 at-
tached to a loadingCompleted notiﬁcation. Note that in
addition to a timestamp, the URI and the title of the docu-
ment that was just loaded, the application gets information
about the user, the process and the view. It also has infor-
mation about the function in the Camino source code that
generated the notiﬁcation.

Instrumenting Camino was both simple and elegant. Patch-
ing its publicly-available code and keeping our version syn-
chronized with the CVS one was quite easy. Finding ob-
servation points inside the code was also relatively easy be-
cause of the Model-View-Controller design pattern followed

fr.lri.insitu.micromegas.camino loadingCompleted
obsPoint: BrowserWrapper.camino loadingCompleted
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. Al-
though about half of our target users have Apple Comput-
ers, 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 sim-
ilar 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. Fire-
fox 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 ex-
tension 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, 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 onunload. 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 extends 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. 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.

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.

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


---

6http://navtracer.mozilla.org/
7http://wiki.mozilla.org/Places

---

8Skype, 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/)


