KEEPING TRACK OF HOW USERS USE CLIENT DEVICES
An Asynchronous Client-Side Event Logger Model

Vagner Figuerêdo de Santana and Maria Cecilia Calani Baranauskas
Institute of Computing, State University of Campinas, Albert Einstein Street, Campinas, Brazil

Keywords: Event logger, Usage logger, Client-side logger.

Abstract: Web Usage Mining usually considers server logs as a data source for collecting patterns of usage data. This solution presents limitations when the goal is to represent how users interact with specific user interface elements, since this approach may not have detailed information about users' actions. This paper presents a model for logging client-side events and an implementation of it as a websites evaluation tool. By using the model presented here, miner systems can capture detailed Web usage data, making possible a fine-grained examination of Web pages usage. In addition, the model can help Human-Computer Interaction practitioners to log client-side events of mobile devices, set-top boxes, Web pages, among other artefacts.

1 INTRODUCTION

Several studies have addressed Web usage, ranging from Usability and Accessibility (A&U) guidelines to tools that analyze code, content, or logs of websites. Additionally, Data Mining is the "analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner" (Hand et al., 2001). The use of Data Mining techniques over Web usage data is called Web Usage Mining (WUM).

WUM algorithms and tools focus mainly on Web server logs. On the one hand, server-side data makes possible to identify the user route in a website requiring less effort, since Web server logs are a natural product of its use. However, server-side logs do not contain representative data about the interactions between the user and the Web page (Etgen and Cantor, 1999). On the other hand, client-side data have more detailed information about user actions in a Web page, but require more effort to capture and transfer the data.

User Interface (UI) events are natural results of using windows based interfaces and their components (e.g., mouse movements, key strokes, mouse clicks, list selection, etc) (Hilbert and Redmiles, 2000). Additionally, event logs produce results as frequency of use of certain functions, places where users spend more time and the sequence that users complete their tasks (Woo and Mori, 2004). Since it is possible to record them and they indicate the user's behaviour, they represent an important source of information regarding usability.

Nowadays, HCI (Human-Computer Interaction) community count on tools that keep track of users behaviour using mouse tracks (e.g., MouseTrack (Arroyo el al., 2006)) and eye tracks (e.g., eyebox2 (Skeen, 2007)), but literature lacks studies focusing on data captured automatically from the whole diversity of users. In particular, accessibility evaluation tools need to address some aspects usually not covered by evaluation tools based on mouse events or visual display. How tools that use mouse or user's eye movements would keep track of screen readers users?

In this context, we present a model to log client-side data and get as many different events as possible, since with a large vocabulary of events, researchers can perform a wider range of analysis. Then a case of study implementation is presented as part of the WELFIT (Web Event Logger and Flow Identification Tool), a tool to identify barriers that assistive technology users face when using websites.

This work is organized as follows: the next section presents works related to client-side events capture; section 3 details the presented model; section 4 discusses some implementation in the Web context; finally, section 5 presents conclusions.

de Santana V. and Calani Baranauskas M. (2009).
KEEPING TRACK OF HOW USERS USE CLIENT DEVICES - An Asynchronous Client-Side Event Logger Model.
In *Proceedings of the 11th International Conference on Enterprise Information Systems - Human-Computer Interaction*, pages 165-168
DOI: 10.5220/0001951001650168

2 CLIENT-SIDE LOGGERS

In this section we will discuss WET (Etgen and Cantor, 1999) and WebRemUSINE (Paganelli and Paternò, 2002) websites evaluation tools based on client-side event logs.

WET focus is on performing automatic capture of events that occur on the client-side, avoiding high costs of time and money present in manual data capture methods (Etgen and Cantor, 1999). The log captured is recorded in text format at client-side (i.e., in cookies) and the capture depends on user actions to capture events. Some of the points that deserve further work involve: some way to record more data, use a bigger event vocabulary, and independence of user actions (Etgen and Cantor, 1999).

WebRemUSINE (Paganelli and Paternò, 2002) makes automatic capture and analysis of websites interaction in order to detect usability problems. The analysis is based on the comparison between the paths made by users and an optimum task model previously configured (Paganelli and Paternò, 2002). The storage and transmission of the logs is done through a Java applet, which allowed the tool to avoid the storage capacity of cookies (Paganelli and Paternò, 2002). For the user, using this tool involves splitting his/her screen into two regions, one for the list of tasks that the participant must choose before each task, and the other containing the website being evaluated (Paganelli and Paternò, 2002).

The common characteristics of these tools result in the main goals of client-side data-loggers: to ***capture*** **events at client-side and to** ***transmit*** **the logged data to a server, where all analysis is made**.

3 THE PROPOSED MODEL

The model was designed so that its set up and use require just one change in the applications to be evaluated: a call to the client-side event logger code. Thus, as soon as a participant starts the test session and accepts to participate on the test, the tool starts to record events occurred at the client-side until the participant cancels his/her participation.

Analysis based on the requirements presented in Santana and Baranauskas (2008) for evaluation tools based on event logs indicated two main components of the model: the **DataLogger**, responsible for capturing event data, and the **Communicator**, responsible for transmitting logs to the server. The **DataLogger** is the component attached to the high level subject to be observed (e.g., Window object), so it can be notified to record all the events occurred. It is inspired on the GoF (Gang of Four) Observer Pattern (Gamma et al., 1995). The **Communicator** component controls the transmission of the logged data to the server. It also keeps the information sent regarding the identification of logs and keeps the server responses. Moreover, other components were added in order to modularize the model and fulfil all the requirements considered.

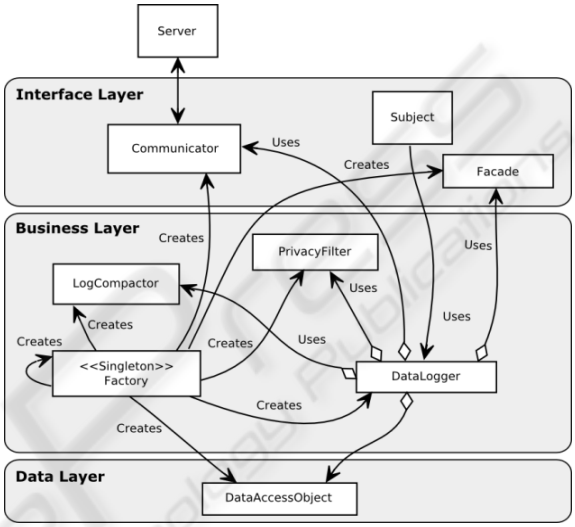

Figure 1: The client-side event logger model overview.

Due to the frequency in which UI events are triggered, the number of events that can occur during few minutes can be huge. Then, any data-logger that must transmit logged data to a server should have to compact the data. This brought the need for a component to compact the data, the **LogCompactor**. It has the role of avoiding the heavy consume of client's bandwidth connection, that may occur if the raw log is transferred to the server. Also, to deal with the amount of data recorded we used asynchronous communication with the server as a strategy to interfere as few as possible with the use of the UI.

To manipulate data and perform record, read, and remove functions, we used a **DataAccessObjec,** which is based on the Data Access Object (DAO) J2EE design pattern (Alur et al., 2003). In addition, we needed a way to interact with the user and show the status of the logger, responsibility of the **Facade** component, inspired on the GoF Facade Pattern (Gamma et al., 1995). To address privacy policies we added the **PrivacyFilter** component, responsible to check if the captured data can or cannot be sent to the server based on previously defined policies (e.g., not record which key is pressed when a keypress

event occur). Finally, we defined a **Factory** to create and assembly all the components together. The **Factory** is the model creator class. It contains the information to instantiate and to build all components. First, the **Factory** instantiates itself, following the GoF Singleton pattern (Gamma et al., 1995), then it uses two other GoF creational patterns to instantiate model classes: Factory Method (Gamma et al., 1995) and Builder (Gamma et al., 1995). The model follows the MVC (Model View Controller) pattern (Figure 1). The main characteristics are:

- It is lightweight and depends on few resources of the users' devices, achieving its goal in different configurations;
- It processes and transmits logs without interfering with the use of the evaluated UI;
- It uses all event available data that do not impact on security problems;
- It logs usage data without depending on specific task models, grammars, or events;
- It provides tool's status and controls, allowing users to interrupt the capture.

4 RESULTS

The implementation of the presented model used JavaScript, an object-based scripting language (Netscape, 1999), and the server module used Java related technologies following the MVC pattern and structure proposed in Basham et al. (2004).

At Web context, the environment configuration of the tool requires that the website administrator registers him/herself and the website to be evaluated, and insert a reference to the JavaScript client-module in the registered website's pages. From that point, if the reference to the client module came from a registered website, then each time an user access the page, the server module fills up the **Factory** component script with information unavailable from JavaScript (e.g., client's IP, a global identifier for that session, etc.) before serving it. Then, as soon as the script is loaded, the tool starts to work.

The JavaScript implementation of the model showed to be efficient and effective. However, there were some issues related to the space available to record information on client's device and to exchange data between different domains. The space available to record information in the client's device is restricted. One solution is to use cookies, but they are limited to a size of 4 kilobytes (kB) and each domain can specify only 20 cookies (Netscape, 1999). When using cookies the problem is the time required to record, retrieve, and delete the logs without interfering with the use of the website. The alternative found was to use the Web page structure in memory, also known as Document Object Model (DOM) tree. Then, application cookies where used only to deal with error recovery and to maintain information valid for more than one session (e.g., the acceptance of the user).

The bigger issue implementing the proposed model in JavaScript was the asynchronous cross-domain transmissions. The problem was to deal with security restrictions of the XMLHTTPRequest, a JavaScript object widely used object in AJAX (Asynchronous JavaScript And XML) applications, which just allows connection between Web pages/applications hosted at the same domain.

The security restriction is called Same Origin Policy and "prevents document or script loaded from one origin from getting or setting properties of a document from a different origin" (Ruderman, 2001). The Same Origin Policy may seem too restrictive, since it blocks the use of Web services directly via XMLHTTPRequests. However, allowing scripts to access any domain opens up users to potential exploitation (Levitt, 2005a).

Some solutions to deal with this restriction are: Signed Scripts (Ruderman, 2007), Server-side proxy, Iframe Proxy (Dojo Toolkit, 2006), and Flash Proxy (Levitt, 2006). These solutions are effective, but they conflict with the requirements we are following, since they are not browser independent, depend on some plug-in or require a more complex environment configuration than presented.

Some proposals that would give to JavaScript programmers the power to perform cross-domain requests are JSONRequest and <module> tag. JSONRequest is proposed to be a new browser service that allows data exchange without exposing users or organization to harm (Crockford, 2006a). The <module> tag proposes to divide a Web page into a collection of modules that are secure from each other, providing safe communication; it also proposes how to reach a consensus on a new Web browser security model, since Web applications are significantly ahead of Web browsers technologies (Crockford, 2006b).

The solution used is based on an approach presented in Levitt (2005b). The approach simulates a JSONRequest using Dynamic Script Tag, which manipulates the DOM tree to perform requests through the creation of script tags, allowing cross-domain asynchronous communication.

Logging tests performed in pages generated by Content Management Systems like Plone and Drupal showed that each second of interaction results in

approximately 1kB of compacted logs. Therefore, any participant using a connection that supports the transmission of 1kB per second plus the average of bandwidth connection used by the participant to surf the Web will allow the model to behave accordingly to the design and does not interfere with the use of the website. If it is not the case, the accumulated amount of log data will reach a configuration limit and the tool will became inactive. The validation of the model was performed capturing events during real use of the website of a research group called *Todos Nós* (www.todosnos.unicamp.br), since part of its audience uses assistive technology. The data captured during 60 days resulted 85 recorded sessions, 6 of them coming from assistive technology users. The data collected resulted in more than 270 thousands of events.

5 CONCLUSIONS

The model proposed showed to be lightweight and addressed requirements stated in Santana and Baranauskas (2008). Also, it can supply data to other applications to discover usage patterns.

During implementation and use of this model, maintainers and developers must always keep security and privacy in mind, since the information being captured and transmitted can be critic if it is not filtered and/or made in a safer way. Accordingly, users must always be aware of what is happening in their device and accept to participate before the logger starts to record events, since the free record of this kind of information without warning the user would characterize the tool as a spyware.

The main advantages of the presented model in comparison with the approaches presented in section 2 are: modularization and configurability of all components, browser and plug-in independent, and compaction, which was not cited in referred works.

Improvements may be obtained through different data compression techniques and transmission plug-in independent approaches allowing the inclusion of security mechanisms.

ACKNOWLEDGEMENTS

Proesp/CAPES and FAPESP.

REFERENCES

Alur, D., Crupi, J. and Malks, D., 2003. Core J2EE Patterns: Best Practices and Design Strategies. 2nd Edition. Prentice Hall PTR.

Arroyo, Ernesto; Selker, T. and Willy, W., 2006. Usability Tool for Analysis of Web Designs Using Mouse Tracks. Work-in-Progress In Proc. of CHI 2006.

Basham, B., Sierra, K., Bates, B. 2004. *Head First Servlets and JSP: Passing the Sun Certified Web Component Developer Exam (SCWCD).* O'Reilly.

Crockford, D., 2006a. JSONRequest. Available at: http://json.org/JSONRequest.html

Crockford, D., 2006b. The <module> Tag. Available at: http://www.json.org/module.html

Dojo Toolkit, 2006. Cross Domain XMLHttpRequest using an IFrame Proxy. Available at: http://dojotoolkit.org/node/87

Etgen, M. and Cantor, J., 1999. What does getting wet (web event-logging tool) mean for web usability? In: Proc. of 5th Conf. on Human Factors & the Web.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J., 1995. *Design Patterns: Elements of Reusable Object Oriented Software.* Reading: Addison Wesley.

Hand, D., Mannila, H. and Smith, P., 2001. Principles of Data Mining. MIT Press.

Hilbert, D.M. and Redmiles, D.F., 2000. Extracting usability information from user interface events. ACM Comput. Surv. 32(4). pp. 384–421.

Levitt, J., 2005a. Fixing AJAX: XMLHttpRequest Considered Harmful. Available at: http://www.xml.com/pub/a/2005/11/09/fixing-ajax-xmlhttprequest-considered-harmful.html

Levitt, J., 2005b. JSON and the Dynamic Script Tag: Easy, XML-less Web Services for JavaScript. Available at: http://www.xml.com/pub/a/2005/12/21/json-dynamic-script-tag.html

Levitt, J.,2006. Flash to the Rescue. Available at: http://www.xml.com/pub/a/2006/06/28/flashxmlhttprequest-proxy-to-the-rescue.html

Netscape Communications Corporation, 1999. *Client-Side JavaScript Reference.*

Paganelli, L. and Paternò, F., 2002. Intelligent analysis of user interactions with web applications. In: IUI '02: Proc. of the 7th Int. Conf. on Intelligent User Interfaces, ACM. pp. 111–118.

Ruderman, J., 2001. The Same Origin Policy. Available at:http://www.mozilla.org/projects/security/components/same-origin.html

Ruderman, J., 2007. Signed Scripts in Mozilla. Available at:http://www.mozilla.org/projects/security/components/signed-scripts.html

Santana, V.F. de and Baranauskas, M.C.C. (2008) A Prospect of Websites Evaluation Tools Based on Event Logs. In IFIP, Volume 272; *Human-Computer Interaction Symposium*; Springer, pp. 99–104.

Skeen, D., 2007. Eye-Tracking Device Lets Billboards Know When You Look at Them. Available at: http://www.wired.com/gadgets/miscellaneous/news/2007/06/eyetracking.

Woo, D. and Mori, J., 2004. Accessibility: A tool for usability evaluation. In APCHI. Volume 3101 of LNCS, Springer. pp. 531–539