Journal of Web Engineering, Vol. 6, No.3 (2007) 244-260


# INSTRUMENTING THE DYNAMIC WEB

ANDY EDMONDS

*Live Search, Microsoft Corp, Redmond, WA*
*andyed@live.com*

RYEN W. WHITE

*Microsoft Research, Redmond, WA*

ryenw@microsoft.com

DAN MORRIS

*Microsoft Research, Redmond, WA*

dan@microsoft.com

STEVEN M. DRUCKER

*Microsoft Live Labs, Redmond, WA*

sdrucker@microsoft.com

Received January 15, 2007
Revised June 6, 2007

One of the most critical driving forces in the evolution of interfaces on the Internet has been the logging built into common Web servers and the decade-long deployment of analytics based upon this data source. Page-view logging has slowly moved to callback systems using client-side scripting to capture more aspects of the user experience. With the rise of JavaScript-based client-side interactivity and, more recently, asynchronous Javascript and XML (AJAX), server-side logging is less able to capture the user experience of Web sites and applications that are rising in complexity. We present a new technique for the in-page logging of interaction events that will help interaction designers make more informed design decisions based on how users are interacting with their systems. The potential benefit of our technique is demonstrated in a case study with a working system.

*Key words*: Usability, analytics, event log analysis, AJAX, interaction design.

## 1 Introduction

The evolution of interfaces on the Internet has been driven by the insights garnered from logging built into common Web servers [1] . Because the protocols used to transfer data on the Web do not provide detailed user interaction information to servers, the focus of the research community has been on studying page-level interactions [2,3]. However, the rise of JavaScript-based client-side interactivity and, more recently, asynchronous Extensible Markup Language (XML) data exchanges have made server-side logging less able to capture the user experience of Web sites and Web-based applications. Modern Web page design is moving user experiences away from historically-typical page transitions

to richer interactions involving background data transfers and client-side content generation within a single Web location. Such complex functionality within the page is used for low-latency input validation, progressive disclosure, dynamic menus, and other multimedia experiences. The increase in complexity of these interfaces and deviation from established interface paradigms makes logging even more critical as a means of determining how this functionality is being employed by end-users. Usage information is of concern to Web site developers and the corporations that rely on such data to inform decisions about page design, marketing emphasis, and even business strategy. To address this emergent dynamism page view logging has recently started to move to callback systems using client-side scripting to capture more aspects of the user experience, such as within-page mouse and keyboard interactions with interface features.

Despite its appeal, logging intra-page activity is an expensive process in a number of ways, including:

- Development cost of instrumenting complex User Interfaces (UIs)
- Potential end-user impact of additional upstream data flow or custom software
- Analysis complexity

Therefore, the Web development community requires a cost-effective technique for logging intra-page activity that minimizes development and deployment cost, minimizes impact on the user, and maximizes impact on the design process. In this article we present *MLogger*, a within-page interaction logging system that incorporates both the latest in browser technology for logging with a design that capitalizes on modern development practices of structural markup and cascading stylesheets. We will explore how detailed in-page logging can improve a user interface design through analysis and design iteration of AJAX-style pages where elements may appear and disappear depending on user input. The logging system is also robust to changes in screen resolution and browser size since tracking is based on page element position rather than absolute pixel location.

In the remainder of this article the range of potential applications and related work will be reviewed, followed by a description of a recent implementation of MLogger. We will then present a case study of the toolkit and a review of other situations in which the toolkit could benefit Web design processes.

## 2 Current Practices and Opportunities

Web development is a complex endeavour with strong business and societal impact. The built-in Hypertext Transfer Protocol (HTTP) server logging of the mid 1990s is rapidly being augmented with client-side callbacks that inform the process of Web development with user data such as browser type, operating system, and screen size, as well as other critical information such as return user tracking (i.e., the frequency with which a user returns to a given site or domain). Many new analytics providers are now trapping the elements that garner clicks and presenting the data in a visualization that starts with the Web page and overlays user activity [4]. For example, Chi *et al*. [5] developed a system that allows users' click behavior as they navigate through the multiple pages of a Web site to be visually analyzed. Improvements to the design of the site can be suggested if appropriate. While this provides an easily-interpretable view of user navigation, it does not address the challenges of Web interaction models which do not revolve around page transitions.

Other systems have allowed developers to study more fine-grained interaction with individual pages within the site. One example of this is the use of "heatmaps" to superimpose cursor attention [6] or eye gaze [7] onto an image of the Web page. Another example is the visualization of eye saccades, gaze fixation, mouse movements, and clicks as a node-link graph allowing chronometric relationships between eye and mouse movements to be identified [8]. Gellner and Forbrig [9] explored the use of custom-installed software to record mouse movements to provide insight into "unconscious aspects of users' behavior" that the authors claim are similar in some respects to body language. However, it is difficult to get from interactions superimposed over a Web page to a detailed analysis of how users are interacting. Additionally, since they are largely based on absolute pixel position, these visualizations are invalidated if the content of the page changes. When users manipulate dynamic content, filter and sort data, or interact in other novel ways within a single page, a new form of logging is needed.

User-Centered Design (UCD) places the user at the center of the design of interactive systems [11]. Performing UCD effectively generally involves up-front user research, iterative testing of prototypes, and – in the age of networked systems – instrumentation and logging to monitor ongoing interface performance. Much of UCD is based on what users say they want rather than what they actually do when confronted with a system. Analyses of log data can complement more traditional UCD methods and provide naturalistic data at scale. From the UCD perspective, top goals for insights from logs span the key dimensions of usability [11]:

- Ease of use
- Error rate
- Usefulness
- Learnability
- Memorability

Robust logs of user activity can provide insights into these user experience dimensions. Techniques such as *sequence detection* (i.e., looking for repeated and semantically-meaningful recurring patterns within a user's activity), *characterization* (i.e., partitioning interaction sessions by the tasks a user attempted), and *comparison* (i.e., generating deviations from an optimal user path for a given intent), along with *multiple levels of coding of events to map to different granularities* (i.e., noting that a user clicked inside a search box but also inside the box's descriptive header) are critical methods for extracting meaning from event logs [12].

Task success is a key ease-of-use metric and is typically identified by iteration between a trigger and an end event for a single session. With the more dynamic UIs of the current Web, both start and end events may be within the same single Web page. However, despite the fact that sequences of interactions have been used previously in log analysis to better understand user behavior [13], they have typically focused on a series of page views, rather than more fine-grained interactions within the pages themselves. Given the recent developments in dynamic UIs, the sequence mining necessary to study task completion may now be needed within a page as opposed to across page loads. For users who start but do not complete the task, sequence characterization can identify key problem points and distinguish between spontaneous *abandonment* (i.e., where users voluntarily terminate interaction activity prior to a recognized stopping point), and error-triggered abandonment (i.e., where users involuntarily terminate interaction activity due to system error). In addition, should an interface have

an affordance for reversing the effect of interactions, the use of this functionality by users following a series of interaction events could help the designer better understand when users are unsatisfied or likely to make mistakes in data entry. To address a variety of aspects of system usability, a system designer can create one or more "optimal paths" [19] that can form the basis for comparison to actual user paths. This can facilitate task identification, ease of use analysis based upon quantitative metrics like number of steps, and the detection of error sequences.

The frequency of usage of an interface feature is a core indicator of usefulness that is also easy to determine given specific UI elements dedicated to triggering features. For example, optional interface features that a designer wishes to study can be hidden by default but made accessible via UI elements on the default display. The frequency with which users request a particular hidden feature is an indicator of its value. In most cases, logging clicks and the elements behind them is sufficient to determine which application features on a page are being used. However, as application state becomes more complex, awareness of a *series* of events may be required to discern the effect of contextually sensitive UI elements. Discriminating between a lack of usefulness and a lack of discovery is a key challenge in mining feature usage.

The use of page content as a feature that should be evaluated for utility is one of the most challenging goals for event log data. A key innovation in the logging framework described in this article is a scheme for logging mouse hover events. Chen [14] suggests that mouse location has moderate predictivity of eye gaze. If user hover time is predictive of user attention, it may be useful to monitor content engagement. In addition, the findings of a number of studies have suggested that page interactivity is predictive of overall satisfaction. Claypool *et al*. [15] showed that time spent on a page and scrolling within a page were predictive of users' self-reported interest in pages across the Web. Fox *et al*. [16] and Joachims *et al.* [17] found similar predictivity in clicks on Web search result pages.

Detecting learnability and relearning of pages and the interface features resident on them requires integrating usage tracking over multiple usage events and brings with it challenges of user recognition over time. For learnability, the lag between page load and user action may be predictive and is enabled by the logging framework proposed here. Mueller and Lockerd [18] report observing highly characteristic mouse movements in cases of high user familiarity. The logging framework described here has not been deployed in a situation of strong re-use, so relearning and memorability will not be addressed directly in this article. However, MLogger does assign a unique identifier to each Web browser that visits the site to help disambiguate new users from return visitors, a key pre-requisite to studying learnability and memorability. Early developments in Web tracking outperformed existing log-based solutions with client-side tracking to more robustly identify individual clients by logging cookie values and to ensure that pages loaded from a browser's cache are captured. In traditional Web applications, an HTTP server typically logs a request for a site's start page with a timestamp and logs a corresponding entry for the end page, if the user got there. In addition to a timestamp, the IP address and browser agent are logged in many default Web server configurations, but these data do not accurately identify individual users or log cached page views.

The MLogger system described here is intended for site developers conducting deep analytic studies of their user interfaces. While our system could be deployed in a usability lab environment, the ability to collect data at scale offers significant advantages in reliability over previous logging systems.

Other approaches to logging enable large-scale Internet research through proxy systems [10] or browser instrumentation [19, 16]. These approaches involve greater user impact, for example software installs or custom browsing portals, than onsite instrumentation but can yield insights beyond the limited purvey of a site owner. Proxies and instrumented browsers are especially useful for understanding behavior across sites, for investigating sites a test team does not control, or for understanding Web usage longitudinally.

These tradeoffs require researchers to consider the goals of their work in order to choose an appropriate method. The rich logging of page details described here is typically only meaningful to the designer as the log captures attributes specific to a page's UI and its implementation. This makes the approach especially useful for site owners, but in fact the technique could also be applied in a proxy or browser shell [19]. The goal in this work is to explore how detailed in-page logging can improve a UI design through analysis and design iteration. In the next section, the logging details are described, followed by a case study of their application.

## 3 Logging System Design

We developed MLogger[a], a logging system that focuses on capturing events within an instrumented Web page with enough detail to programmatically recreate a user's experience and to describe the user's interactions in the language of the site designer. A key challenge in production-ready event logging is browser compatibility. While Microsoft's Internet Explorer (MSIE)[b] still has the majority of the market share, some of the most active Web users are using Mozilla's Firefox[c] or other alternative browsers. To address the complexities in cross-browser development, and to ease the tasks of Web application construction, a multitude of robust client-side libraries are available for dynamic Web application development. These libraries simplify the cross-browser differences and facilitate asynchronous data exchange, hiding and showing content, animation, and general presentation functions. MLogger uses MochiKit[d] to support MSIE and Firefox. The event monitoring libraries that MochiKit provides are compatible with standards proposed by Microsoft and the World Wide Web Consortium (W3C). In addition, the library not only provides a single way to invoke event listeners but also normalizes the returned metadata.

### *3.1 Event Listeners*

Trapping user events with JavaScript is accomplished by attaching functions, called "listeners," to combinations of the Document Object Model (DOM) and user actions. The following user actions, or "events," have a logging function attached to them at the document level.

MLogger exploits the Web page's DOM and common best practices for constructing dynamic application pages.

These practices state that presentation aspects be attached to HTML with cascading style sheets

[a] The MLogger name is derived from "MochiKit Logger".
[b] http://www.microsoft.com/ie
[c] http://www.firefox.com
[d] http://mochikit.com

(CSS) effectively identifying repeated classes of UI elements with a common identifier. Further, key nodes in the page are tagged with unique identifier (ID) attributes to facilitate addressing those nodes with scripting code. Through these methods a highly descriptive and detailed event log can be constructed containing instances of the events described in Table 1.

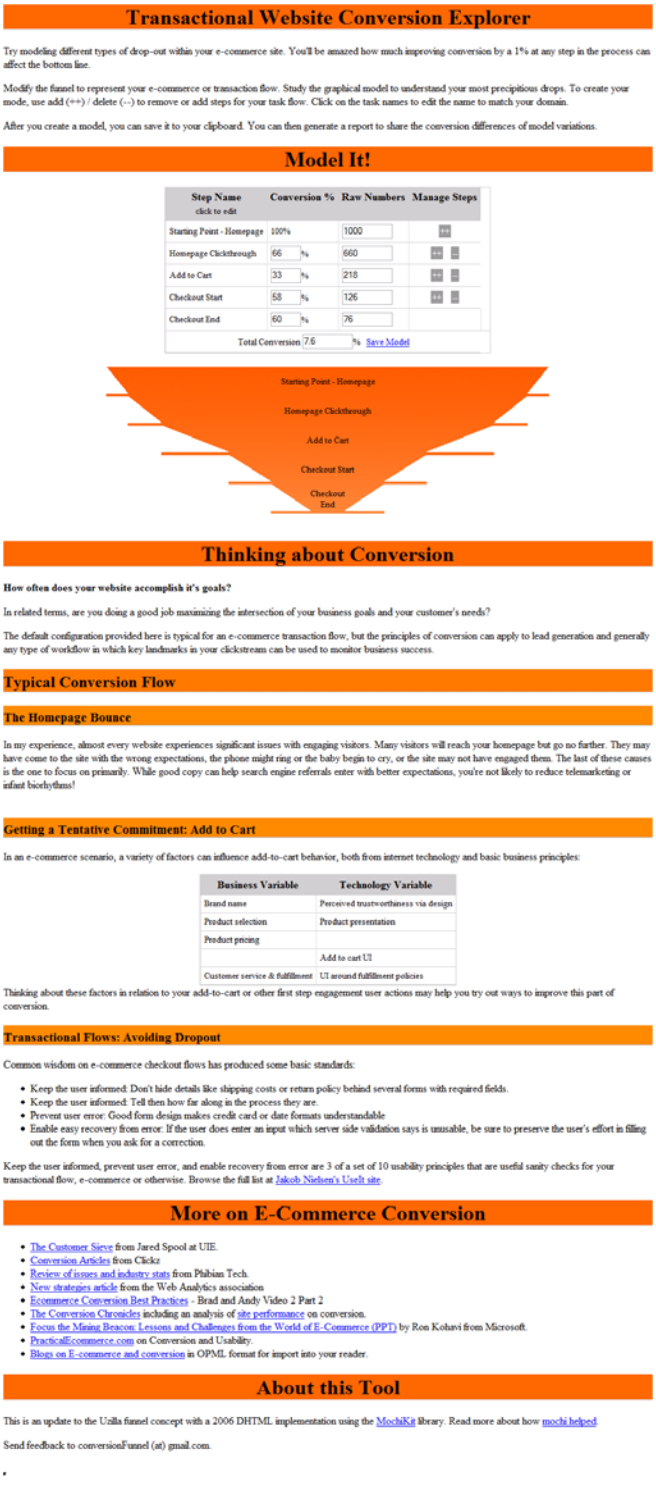
Table 1. User actions or events captured in MLogger.

| Event | Log frequency |
|---|---|
| Mouse click | All |
| Mouse movement | If underlying element is different than last move |
| Key press | All |
| Scroll | All (time since last scroll event + duration) |
| Page leaves focus (blur) | All |
| Page gets focus | All |
| Window resize | All |

In the next section we describe a case study in which we deployed MLogger on a Web site targeted at Web analytics community, recruited subjects from this community, and logged and analyzed their interactions.

## 4 A Case Study Using MLogger

We developed a Web-based application to help Web analysts construct predictive models of consumer behavior on e-commerce Web sites. We used MLogger to record user interactions with this application as defined earlier in Table 1. To test the performance of the system we recruited subjects from two usability-focused Weblogs and an analytics mailing list. Subjects were encouraged to visit the page on which the application was hosted and use the facilities it offered in whatever way they saw fit. It is important to note that we did not give users an explicit task to perform with this system. The aim of this exercise was to gather a fairly large amount of usage data with the application, and test the robustness of MLogger at storing and analyzing the interaction data that resulted. Nonetheless it was assumed that the user population targeted in the recruitment email would be interested in using this application for its intended purpose of determining Web site conversion rates, so the data we would get would be fairly representative of the type of logs that would be generated by this demographic.



# Transactional Website Conversion Explorer

Try modeling different types of drop-out within your e-commerce site. You'll be amazed how much improving conversion by a 1% at any step in the process can affect the bottom line.

Modify the funnel to represent your e-commerce or transaction flow. Study the graphical model to understand your most precipitious drops. To create your mode, use add (++) / delete (--) to remove or add steps for your task flow. Click on the task names to edit the name to match your domain.

After you create a model, you can save it to your clipboard. You can then generate a report to share the conversion differences of model variations.

## Model It!

| Step Name (click to edit) | Conversion % | Raw Numbers | Manage Steps |
|---|---|---|---|
| Starting Point - Homepage | 100% | 1000 | ++ |
| Homepage Clickthrough | 66 % | 660 | ++ -- |
| Add to Cart | 33 % | 218 | ++ -- |
| Checkout Start | 58 % | 126 | ++ -- |
| Checkout End | 60 % | 76 | |
| Total Conversion | 7.6 % | Save Model | |

Starting Point - Homepage
Homepage Clickthrough
Add to Cart
Checkout Start
Checkout End

## Thinking about Conversion

**How often does your website accomplish it's goals?**

In related terms, are you doing a good job maximizing the intersection of your business goals and your customer's needs?

The default configuration provided here is typical for an e-commerce transaction flow, but the principles of conversion can apply to lead generation and generally any type of workflow in which key landmarks in your clickstream can be used to monitor business success.

### Typical Conversion Flow

#### The Homepage Bounce

In my experience, almost every website experiences significant issues with engaging visitors. Many visitors will reach your homepage but go no further. They may have come to the site with the wrong expectations, the phone might ring or the baby begin to cry, or the site may not have engaged them. The last of these causes is the one to focus on primarily. While good copy can help search engine referrals enter with better expectations, you're not likely to reduce telemarketing or infant biorhythms!

#### Getting a Tentative Commitment: Add to Cart

In an e-commerce scenario, a variety of factors can influence add-to-cart behavior, both from internet technology and basic business principles:

| Business Variable | Technology Variable |
|---|---|
| Brand name | Perceived trustworthiness via design |
| Product selection | Product presentation |
| Product pricing | |
| | Add to cart UI |
| Customer service & fulfillment | UI around fulfillment policies |

Thinking about these factors in relation to your add-to-cart or other first step engagement user actions may help you try out ways to improve this part of conversion.

#### Transactional Flows: Avoiding Dropout

Common wisdom on e-commerce checkout flows has produced some basic standards:

- Keep the user informed: Don't hide details like shipping costs or return policy behind several forms with required fields.
- Keep the user informed: Tell then how far along in the process they are.
- Prevent user error: Good form design makes credit card or date formats understandable
- Enable easy recovery from error: If the user does enter an input which server side validation says is unusable, be sure to preserve the user's effort in filling out the form when you ask for a correction.

Keep the user informed, prevent user error, and enable recovery from error are 3 of a set of 10 usability principles that are useful sanity checks for your transactional flow, e-commerce or otherwise. Browse the full list at Jakob Nielsen's Useit site.

## More on E-Commerce Conversion

- The Customer Sieve from Jared Spool at UIE.
- Conversion Articles from Clickz
- Review: of issues and industry stats from Phibian Tech.
- New strategies article from the Web Analytics association
- Ecommerce Conversion Best Practices - Brad and Andy Video 2 Part 2
- The Conversion Chronicles including an analysis of site performance on conversion.
- Focus the Mining Beacon: Lessons and Challenges from the World of E-Commerce (PPT) by Ron Kohavi from Microsoft.
- PracticalEcommerce.com on Conversion and Usability.
- Blogs on E-commerce and conversion in OPML format for import into your reader.

## About this Tool

This is an update to the Uzilla funnel concept with a 2006 DHTML implementation using the MochiKit library. Read more about how mochi helped.

Send feedback to conversionFunnel (at) gmail.com.

Figure 1. Funnel Application

Over the course of two days, 67 users visited the page and their usage was captured by MLogger. We did not record any personal information about users, nor did we request that they complete any questionnaires about their experience prior to departure from the Web site. The only information we recorded was users' interactions with the application.

### *4.1 Sample Application*

The single-page application was constructed using client-side JavaScript for rich interactivity specifically to test the logging framework. Figure 1 shows the page containing the application, featuring a spreadsheet-like calculator at the top with a parallel graphic rendering of the data. The page models an e-commerce flow from "home page" to "add to cart" and through the checkout process. Users of the Web site can explore the impact of varying completion rates at each stage. Additionally, support for custom steps is present but expected by the page designer to be less frequently used. We refer to this as the "funnel" application, since its aim is to estimate eventual conversion rates from an initial user pool and generate a summary report that site designers can use in developing their e-commerce application.

The spreadsheet grid, pictured below, allows editing of either percent or counts as well as the addition of new rows. Clicking a step name turns the cell into an editable text field, allowing it to be renamed. The "manage steps" column supports creating additional rows or deleting existing rows.

| Step Name<br>click to edit | Conversion % | Raw Numbers | Manage Steps |
|---|---|---|---|
| Starting Point - Homepage | 100% | 1000 | ++ |
| Homepage Clickthrough | 66 % | 660 | ++ – |
| Add to Cart | 33 % | 218 | ++ – |
| Checkout Start | 58 % | 126 | ++ – |
| Checkout End | 60 % | 76 | |
| Total Conversion | 7.6 % | Save Model | |

Figure 2. Spreadsheet grid in funnel application

This example offers the Web designer a chance to explore task success. We will assume that complete user success is defined, by the site designer, as manipulating the spreadsheet to customize one or more models, saving the models, and then generating a final report. The spreadsheet features provide an interaction design challenge for representing non-standard UI elements on the Web, e.g., adding rows to a table and editing text. The supporting text offers an opportunity to explore measurement of user interaction with page content rather than only interface elements such as text boxes and buttons.

*4.2 Identifying Unique and Repeated Elements*

Well-designed structural HTML uses ID attributes to identify key unique elements on the page and CSS classes to identify repeating elements. For understanding user behavior, this distinction is very useful. Storing CSS class names allow aggregation of data across repeated UI elements. This is particularly useful for assessing interaction design issues. Unique element identification is more useful for task model and state transition analyses along with specific data mining.

The spreadsheet grid shown in Figure 2 has several sets of repeating elements: the editable "Step Name" cells, the "Conversion %" fields, the "Raw Number" input fields, and the "Manage Steps" buttons. Individual rows are labelled with ID attributes indicating their offset, and this offset is managed dynamically as new rows are added.

Events in the browser target the most nested element in the DOM. Meaningful description of a UI element may occur in the ID or class attributes higher in the DOM. To address this, the logging system walks up the DOM from the event element to the root node (the document body) [19]. In addition to node attributes, the node tag name and offset of the tag within the parent block is recorded to enable precise and programmatic identification. For example, an event occurring in a typical site search box might occur within a form of id "searchForm" enclosed in the DIV identified by the attribute "header" which is a child node of the BODY tag.

The DOM path column lists the ID, class, and node name and offset for each step up the DOM tree to the body. Table 2 demonstrates the level of detail available with each logged event in the funnel application; the element ID's and classes have specific meaning to the page's designer that identify an element's role in the UI.

Table 2. Sample mouse move event as represented in MLogger

<table>
<tr><th colspan="2">Sample Mouse Move Event</th></tr>
<tr><td>DOM Path</td><td>
<table>
<tr><th>Element ID</th><th>Element Class</th><th>Element Offset</th></tr>
<tr><td></td><td>dataNumber</td><td>TD(2);</td></tr>
<tr><td>dataRow2</td><td>dataRow</td><td>TR(2);</td></tr>
<tr><td>conversionTableBody</td><td></td><td>TBODY(0);</td></tr>
<tr><td>conversionTable</td><td>conversionTable</td><td>TABLE(0);</td></tr>
<tr><td></td><td></td><td>FORM(0);</td></tr>
<tr><td>conversionForm</td><td></td><td>DIV(0);</td></tr>
</table>
</td></tr>
<tr><td>Timestamp</td><td>1529 milliseconds from page render</td></tr>
<tr><td>Type</td><td>mousemove</td></tr>
<tr><td>X</td><td>748 pixels</td></tr>
<tr><td>Y</td><td>381 pixels</td></tr>
</table>

This demonstrates how the logging in MLogger can be tailored to the interface being analyzed, allowing more accurate descriptions of interaction events to be captured. The DOM element logging enables programmatic recreation of the user event sequence. This offers significant advantages over video logs that might be collected in a usability lab, where this data would have to be coded at enormous time and expense and potential inaccuracy. While exact mouse position is not recorded continuously (in the interests of keeping the load on the system to a minimum and the size of the log file down), mouse position is logged with each mouse action that results in a new underlying DOM element being hovered. This also enables a visualization that is independent of screen resolution, which may differ across users, making Mlogger potentially more portable than existing logging technologies.

The design of the logging details has a strong effect on the size of the data, burdening the client and the analyzer. It is easy to toggle the level of detail, and even the DOM path approach can be customized to a particular needed level of detail. Logging mouse hover locations only when the underlying page structure changes is a verbose but strategically limited level of logging. Logging every mouse movement event, for instance, can generate ten or more events per second.

While the DOM offset is the most precise way to enable event playback, and is often critical with dynamic applications, it is sensitive to small changes in the HTML mark-up. The ID-based approach is more resilient to redesigns and is the likely source of comparative analyses through redesigns. The offset-based approach is also useful for identifying elements in log data even if they lack an ID attribute. Both class names and ID's provide useful data for sequence mining user event logs.

Several additional data points are logged, depending upon the type of an event. Table 3 details these additional attributes.

Table 3. Data points logged

| **Event Type** | **Additional Information** |
|---|---|
| Mouse Move | DOM Path |
| Resize | Screen width, screen height |
| Link Click | HREF attribute of link |
| Scroll | Y-offset |
| Key-presses | Key character code and lookup table identifier |
| Blur / Focus | Toggle flag indicating current focus |

Different events carry different metadata and change different aspects of the UI state. For example, clicking a link typically does not change the y-offset, so logging this attribute is not efficient. Right-clicks and drag-drop events were not of interest for the current page studied but could be added. In the next section we describe the analysis of the funnel application.

## 5 Analyzing the Funnel Application

### *5.1 Task Success*

For many Web designers, our toolkit allows a detailed investigation of *task success*: the progress of a user through a series of specific interactions that the designer wants users to complete. For example, while our example page was built to construct models of conversion flow and generate a summary report. The user's initiation of this process is identified by a click in the any of the data grid form elements; we can detect this event in our logs by selecting clicks in any DOM path that contain the class "dataNumber" or "dataPercent". The end of this process is identified by a click on the "generate report" link; we can detect this event in our logs by selecting clicks that contain the ID "generateReportLink". This analysis is typical of how we expect Web designers to use our toolkit; generating a "did not attempt" rate and an "attempted but did not complete" rate (similar to the notion of abandonment described earlier) is essential to evaluating the success of nearly all commercial Web applications.

### *5.2 Errors*

Two core error conditions exist in almost any commercial Web application: the "did not attempt" and "attempted but did not complete" conditions. Identification of errors and stop conditions is typically a data-mining challenge. One exception is a pervasive task failure condition for almost all Web pages: the "homepage bounce". In this scenario, the user chooses not to engage with the page immediately following a basic inspection. In our example, given that our funnel application is a single page, the equivalent failure condition is a page view without scrolling or form interaction. Analysis of user action showed that 44% of users engaged with the page. The bounce rate for this application (i.e., the percentage of visitors who leave the page without explicitly interacting with the applicaton) , computed by looking at the event traces where an exit event was logged without any scrolling or less than 3 hover locations. Given the opt-in nature of participation, this reflects both the UI and the method of soliciting users. Importantly, traditional Web logging systems would not allow a designer or site administrator to compute these figures, since the condition for engagement does not require additional page requests. In Table 4 we show the percentage of users who engaged with the application for different types of engagement and different tasks.

Table 4. User engagement and associated tasks for our example application

| **User Engagement** | **Percent of Users** | **Task** |
|---|---|---|
| Scroll or Click in Spreadsheet UI | 43.75% | All |
| Any Spreadsheet Click | 29.68% | Spreadsheet |
| Model Editing | 18.75% | Spreadsheet |
| Scroll | 17.18% | Learn |
| Generate Report | 3% | Spreadsheet |

One can look at this page as supporting two distinct tasks: learning about e-commerce conversion from the text ("Learn" in Table 4), and links and exploring specific conversion models ("Spreadsheet" in Table 4). The percent of users that scroll (17%) is indicative of the percentage of users that are interested in learning about e-commerce conversion. The percentage of users that edit spreadsheet cells (18%) is indicative of the percentage of users that are interested in exploring specific conversion models. These metrics of user interest are again based on client-side activity and could not be computed using traditional logs gathered during HTTP request events.

Analysis of low-level user errors is another area of significant interest to all Web designers that cannot be addressed using traditional page-based logging. Dragging and dropping page objects to an invalid location, submitting an invalid password, and a multitude of other everyday slips affect computer usage. Many such errors can be expected in advance and are either allowed to occur due to resource limitations or are handled explicitly by page logic, but error conditions that are *not* anticipated by the page designer are virtually impossible to detect using traditional request logs. MLogger allows post-hoc analysis of interaction data to identify and handle such errors by identifying an event signature and mining the data stream around such occurrences.

In addition to this, MLogger also allows for the visualization of user interaction with elements of the interface. For example, Figure 3 shows a heatmap depicting the percentage of users who hovered over the elements in the spreadsheet form in our sample application.

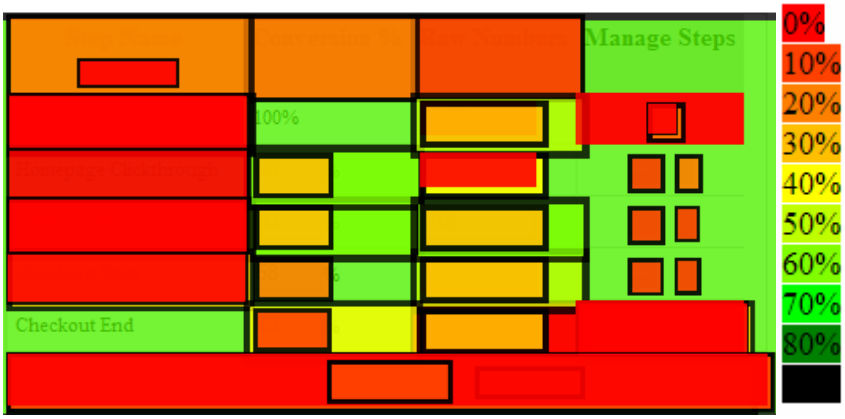



Figure 3. Heatmap of percent of users hovering on the spreadsheet form in our sample application. This visualization could not be generated using traditional logging approaches

In the visualization presented in Figure 3, color represents the percentage of users hovering over the various elements in the grid. A Web designer reviewing this data might, for example, look for *non-interactive* areas over which users hover, potentially indicating user confusion. In this example, the 30% hover rate in the upper-left-most cell is interesting as the content of that cell is "Model Name: click to edit". A Web designer might therefore further examine this scenario by querying the data log for mouse clicks in this cell, potentially indicating a misinterpretation by the user of the instructions. In this case, no user clicked on the instructional text, likely indicating that users hovered here to

consider the instructional text or that this element was simply close to the center of the page, leading to frequent mouse dwells.

Also noteworthy is the resolution-independent nature of this visualization, made possible by our DOM-based logging scheme. Since our heat map is based upon the underlying page representation, differences in mouse position are normalized across screen sizes. The density of hover is computed at each level of the DOM: at the table, row, cell, and cell content levels.

In this section we have presented some analysis of the findings of our case study to demonstrate what can be done with the MLogger application. The granularity of the analysis and the low overhead on systems is obviously appealing to the designers of commercial Web sites, accessed by many users on a daily basis. In the next section we present an example of how this logging could benefit a highly interactive commercial application.

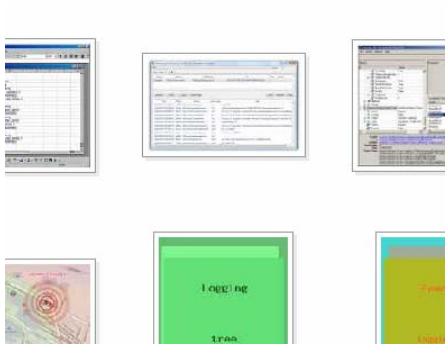
## 6 A Consumer Application of Mouse Hover Events

Windows Live Image Search[e] is one of a growing set of next-generation user interfaces in which page transitions have been minimized. We discuss this page here as an example of a system that is difficult to study with traditional logging methods, but can be examined in detail using MLogger.

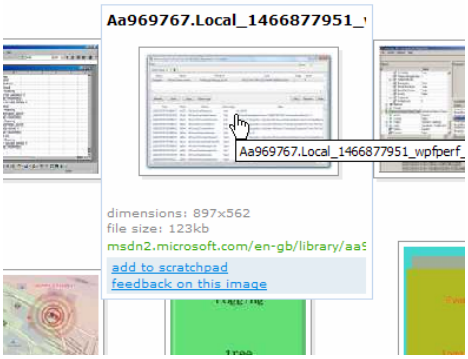
In Live Image Search, hovering over an image triggers an image zoom, reducing the utility of "more detail" pages. Background data transfers eliminate "next page" operations (the interface implements a concept of "infinite scrolling", so there is no need for the "next" button commonly available in Web search interfaces). Dynamic filters allow direct control over the size of the images shown, and therefore

Figure 4. A Potential Error Condition by Over-shooting a Popup Affordance

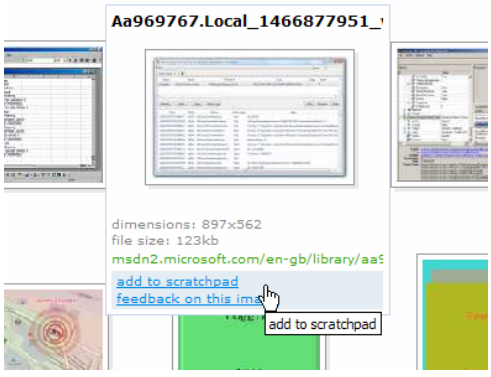
Step 1: Prior to hover



Step 2: Image hover




Step 3: Accidental popup triggered by mouse overshoot




[e] http://images.live.com

the number of images on the page. The interface also offers a "scratchpad" into which users can drag and drop images for later examination (not shown in Figure 4).

Hovering over any image brings up more details about that image, including the dimensions, the file size, the location, and additional options. Pictured to the right is a successful user interaction with a hover detail view, revealing additional affordances. Step 1 pictured the default view, step 2 the hover, and step 3 the successful selection of a hyperlink at the bottom of the popup.

A heuristic review of the hover affordance resulted in the concern that the user might overshoot the bottom of the hover popup and trigger the popup on the image below it. If events were to be logged at the page level then we could not capture such erroneous interactions. However, instrumenting mouse hover events with summaries of the underlying DOM would reveal whether this hypothesized issue actually affects end users.

Since search engine engagement can be an indicator of user interest [20] the lack of normal paging events necessitated sub-page level instrumentation for this application. As a result, user interaction data is being studied to drive future UI iterations of this application.

## 7 Challenges in Instrumenting Web Applications

### *7.1. System State*

A key contributor to the commercial success of Web-based analytics has been the stateless nature of the Web. As more complex applications are built for the Web, the URL to which a user has navigated will no longer be a useful indicator of system state. Identifying states for task analysis, user experience playback, and error conditions will thus become more difficult. While the framework discussed here does allow events to be played back to recreate page state, this makes cross-user visualizations more difficult and adds a layer of complexity to sequential analysis for pattern identification. Analysis routines have to maintain an ongoing record of the state of the UI in order to effectively group events to a task model.

### *7.2. Data Transport*

The system described here creates a slight delay in the time at the point the user selects a new page to load, closes the window, etc. in order to send data to the server. The specific event handler is known as "onbeforeunload" and triggers when a URL change occurs. While this is a common event logging technique in commercial systems, the data size here is larger than typical, occasionally exceeding twenty kilobytes in an uncompressed format with a greater user experience impact. Other systems use proxy servers [21] or instrumented browsers [15] to reduce the impact of data travelling up to the server, or use local file resources

If an application is dynamic (fetches data in the background without page refresh), data logging should be bundled with these requests for data. In the case study here, the exit event is used as a trigger to send data to the server. Additionally, a pause of greater than 20 seconds also triggers a send to ensure capture of more abrupt browser process termination. Data is stored on the client in a Javascript object and serialized to XML for sending to the server via form post. Form-get, a less

cumbersome transport mechanism using purely the location string, is preferable but would require a more concise serialization strategy than the one used here due to limits in the size of URLs.

### *7.3 Data Analysis and Interpretation*

Analyzing event streams is difficult, particularly when event streams are collected at Web scale. Tools and techniques for identifying sequences, sub-characterizing within those sets, and looking for correlations between activity patterns are needed. A particular challenge in measuring task success is identifying when a user *begins* working on a particular task. This is especially difficult following the transition from a previous task. The boundaries between tasks can be fuzzy and multitasking is a common operation [22, 23]. More work is necessary in determining when new tasks begin. Research on task switching is ongoing in the human-computer interaction and artificial intelligence communities [24, 25]. User interaction logging of the type proposed in this paper can play a part in better understanding how users use Web applications and transition among tasks.

## 8 Future Directions

### *8.1 Methods for Content Assessment*

There is reason to hypothesize that within -page activity could inform Web designers about the quality of the content on a page. Certainly, page interactivity is predictive of overall satisfaction. Claypool *et al*. [15], showed that time spent on page and scrolling were predictive of users' self-reported interest in pages across the Web.

There is, however, little existing work on measuring engagement with the page directly. Claypool found that a simple click count did not predict interest, but this study was conducted more than five years ago when interactive applications were less common. The utility of eye tracking for understanding Web usability [8] is well demonstrated, but there is little scientific work on assessing content purely through event logging.

Our system includes several data streams that provide information about page content. One hypothesis, supported somewhat by Mueller, and Lockerd [18] and Chen and Anderson [14], is that a user's mouse tends to follow his gaze, and an estimate of a user's gaze position can be used to estimate interest in various page components. As suggested earlier, Mueller and Lockerd found 30% of users kept their mouse in synch with their eyes while Chen and Anderson reported more complex relationships between eye and mouse locations.

A lack of scrolling has been reported above, but there is an interesting pattern of scrolling and then revisiting already viewed portions of the page. Backtracking is a well-known phenomenon in reading and can be correlated with a user's working memory load and with error states requiring revisitation of key content. More global revisits, where large portions of the page are reviewed, are likely a more positive sign, given the tendency of Web readers to scan and selectively consume content

### *8.2 Measuring User Satisfaction*

While the scenarios here have shown strong use cases for event log data, a more general notion of satisfaction is useful in the process of producing and evolving interface design. Particularly for

applications or complex Web sites, satisfaction is a complex outcome variable that encompasses all of Nielsen's dimensions of usability [11]. Creating a mapping from logged data to satisfaction is often challenging. Combining event logging and explicit user feedback enables an application-dependent effort to predict satisfaction from event logs. In Fox *et al.* [16], a lightweight, explicit feedback collection effort combined with event logging produced a reusable model that will predict the overall satisfaction outcome variable from purely event logs – at least until the application changes enough to reduce the validity of the model. This can be used to address a key challenge in the use of event logs: interpretation. While breaking down the potential tasks a user might engage in on a Web application facilitates segmenting user behavior into comparable sets, a machine learning approach ignoring task variations can capture powerful regularities in behavior logs predicting user satisfaction across user intents.

## 9 Conclusions and Future Work

Now that Web applications are reaching the complexity of traditional desktop software, it is possible that the success of analytics on the Web will catalyze the deployment of rich instrumentation to the desktop. In any event, the extreme competition and low switch costs for a given service on the Web, will create an increasing opportunity for interface innovation, and are likely to make the form of detailed logging described here a topic of great utility. On the research side, the pervasiveness of Web UIs are increasing the appeal of studying browser based applications.

We described MLogger, a logging application that both facilitates user activity playback in dynamic UIs and analyses meaningful to designers by capturing a rich representation of the events in the UI at multiple levels of granularity. Modern Web development best practices use a practical separation between content and presentation style, creating a hierarchy of repeated design elements upon which log data may be aggregated. Measuring task success in rich internet applications (RIAs) was demonstrated, as was quantifying error conditions and assessing feature usage versus feature discovery. As applications are becoming increasingly common on the Web, measuring content engagement and overall user satisfaction may be facilitated by this type of low-level logging.

## References

1. Kohavi, R. (2001). Mining e-commerce data: the good, the bad, and the ugly. In *Proceedings of the 7th ACM SIGKDD international Conference on Knowledge Discovery and Data Mining*, 8-13.
2. Catledge, L.D., and Pitkow, J.E. (1995). Characterizing browsing strategies in the World Wide Web. *Computer Networks and ISDN Systems*, 27(6): 1065-1073.
3. Huberman, B., Pirolli, P., Pitkow, J., and Lukose, R. (1998). Strong regularities in World Wide Web surfing. *Science*, 280(5360): 95–97.
4. Peterson, E. (2006). Web Analytics Demystified: *A Marketer's Guide to Understanding How Your Web Site Affects Your Business*. Celilo Group Media: Portland.
5. Chi, E.H., Pirolli, P., and Pitkow, J. (2000). The scent of a site: a system for analyzing and predicting information scent, usage, and usability of a Web site. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 161-168.
6. Weinreich, H., Obendorf, H., Herder, E., and Mayer, M. (2006). Off the beaten tracks: Exploring three aspects of Web navigation. In *Proceedings of the World Wide Web Conference*, 133-142.
7. Cutrell, E. and Guan, Z. (2007). What are you looking for? An eye-tracking study of information usage in Web search. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (in press).

8. Reeder, R.W., Pirolli, P., and Card, S.K. (2001). WebEyeMapper and WebLogger: Tools for analyzing eye tracking data collected in Web-use studies. In *Extended Abstracts of the ACM SIGCHI Conference on Human Factors in Computing Systems*, 19-20.
9. Gellner, M. and Forbrig, F. (2003). ObSys: A tool for visualizing usability evaluation patterns with mousemaps. In *Proceedings of HCI International*, 469-473.
10. Atterer, R., Wnuk, M., and Schmidt, A. (2006). Knowing the user's every move: User activity tracking for Website usability evaluation and implicit interaction. In *Proceedings of the World Wide Web Conference*, 203-212.
11. Nielsen, J. (1990). *Hypertext and Hypermedia*. Academic Press.
12. Hilbert, D. and Redmiles, D. (2001). Large-scale collection of usage data to inform design. In *Proceedings of 8th IFIP TC 13 Conference on Human-Computer Interaction*, 569-576.
13. Pitkow, J. and Pirolli, P. (1999). Mining longest repeating subsequences to predict World Wide Web surfing. In *Proceedings of the USENIX Symposium*, 139-150.
14. Chen, M.C., Anderson, J.R., Sohn, M.H. (2001). What can a mouse cursor tell us more? Correlation of eye/mouse movements on Web browsing. In *Extended Abstracts of the ACM SIGCHI Conference on Human Factors in Computing Systems*, 281-282.
15. Claypool, M., Le, P., Waseda, M., and Brown, D. (2001). Implicit interest indicators. In *Proceedings of the Conference on Intelligent User Interfaces*, 33-40.
16. Fox, S., Karnawat, K., Mydland, M., Dumais, S.T., and White, T. (2005). Evaluating implicit measures to improve the search experience. *ACM Transactions on Information Systems*, 23(2): 147-168.
17. Joachims, T., Granka, L., Pan, B., Hembrooke, H., and Gay, G. (2005). Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th Annual ACM SIGIR Conference on Research and Development in Information Retrieval*, 154-161.
18. Lockerd, A. and Mueller, F. (2001). Cheese: Tracking mouse movements on websites. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing* Systems, 279-280.
19. Edmonds, A. (2003). Uzilla: A new tool for Web usability testing. *Behavior Research Methods, Instrumentation and Computers*, 32(2): 194-201.
20. Agichtein, E., Brill, E., Dumais, S.T., and Ragno, R. (2006). Learning user interaction models for predicting Web search preferences. In *Proceedings of the 29th Annual ACM SIGIR Conference on Research and Development in Information Retrieval*, 3-10.
21. Hong, J. and Landay, J. (2001). WebQuilt: A proxy-based approach to remote Web usability testing. *ACM Transactions on Information Systems*, 19(3): 263-285.
22. Miyata, Y. and Norman, D.A. (1986). Psychological issues in support of multiple activities. *In* Norman, D. A. and Draper, S.W. (Eds.) *User Centered Design, Lawrence Erlbaum*, 265-284.
23. Rubenstein, J.S., Meyer, D.E, and Evans, J.E. (2001). Executive control of cognitive processes in task switching. *Journal of Experimental Psychology*, 27(4): 763-797.
24. Czerwinski, M., Horvitz, E., and Wilhite, S. (2004). A diary study of task switching and interruptions. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, 175-182.
25. Horvitz, E., Kadie, C.M., Paek, T., and Hovel, D. (2003). Models of attention in computing and communication: From principles to applications. *Communications of the ACM*, 46(3): 52-59.