Data Collection Methodologies for Web-Based Experiments

Stu Westin
University of Rhode Island, USA

INTRODUCTION

Experimental studies involving the use of the World Wide Web (WWW) are becoming increasingly common in disciplines such as management information systems (MIS), marketing, and e-commerce. The focus of these studies is varied and may involve issues of human factors and interface design (Otto et al., 2000; Koufaris, 2002; Liang & Lai, 2002; Palmer, 2002), issues of information processing and search strategies (Spence, 1999; Johnson et al., 2000; Xia & Sudharshan, 2000; Chiang et al., 2004), issues of vendor trustworthiness (Grazioli & Jarvenpaa, 2000; Jarvenpaa et al., 2000; Norberg, 2003), or a myriad of other topics. Regardless of the issue being studied, data collection for online Web research often proves to be a vexing problem, and ideal research designs are frequently sacrificed in the interest of finding a reasonable data capture mechanism. In this article, we discuss some of the methodological complexities that arise when conducting Web-based experiments. We then describe an innovative, software-based methodology that addresses these problems.

BACKGROUND

Despite the difficulties involved, the research community has recognized the importance of Web-based experimental research (Saeed et al., 2003; Gao, 2003).

Server-side data collection approaches based on active server page (ASP) scripts or the like can prove useful in some circumstances. Consider, for example, the situation where one wants to investigate the impact of download time. This has been studied, for example, as a factor affecting user satisfaction (Otto et al., 2000), and Web site success (Palmer, 2002). Using ASP scripts, a delay mechanism can be easily built into a Web page so that the server will delay serving the requested page to the client until some precise, predetermined time has passed. Different experimental treatment levels are accomplished by merely manipulating the delay time that is scripted into the Web page. Here, the experimental subject, using an ordinary browser, will have the perception that the page is slow to download because of the delay between when the page is requested (e.g., by clicking a hyperlink) and when the page is available in the browser.

As another scenario, consider the situation where the researcher wants to study the end user's Web search strategy (e.g., Jansen et al., 2000; Johnson et al., 2000; Chiang et al., 2004) by considering which pages are accessed, along with the sequence of page access. In this case, we can analyze the so-called *click-stream data*. The research community has, in fact, called on investigators to exploit "non-intrusive means of collecting usage and exploration data through click streams" (Gao, 2003, p. 31) in future Web studies. Here again, ASP scripts in the Web pages can provide a simple data collection mechanism by logging each page request (page ID, server time stamp) in a server database. In both of the above research scenarios, standard Web browser software such as Internet Explorer (IE) can be used in the experiment.

In considering these research situations, it is obvious that client-side data collection mechanisms can be constructed just as easily. In both cases, Java applets, Java scripts, or VB scripts can be embedded into the HTML pages to handle the required tasks, and, again, standard browser software can be used. The only difference in this client-side approach is that the data collection is being handled by the client rather than by the server machine. Neither approach provides any obvious benefits over the other, although in the client-side approach, the Web pages could be stored locally, and thus WWW, or even network access, is not required.

One flaw in all of these research scenarios lies in the fact that experimental access must be restricted to a limited set of Web pages that have been appropriately scripted for data collection. If the experimental subject is allowed to "wander" beyond this limited set of pages (an activity that is fundamental to the nature of the Web), then these actions will not be recorded, and the validity of the experiment will be nullified; there will simply be no data collection scripts to execute. Related to this is the fact that all Web pages used in the experiment must be developed and maintained by the investigator—a task that can be labor intensive if a large number of pages are to be made available. Obviously, the experimental pages should be large in number and professional in appearance if external validity is to be maintained.

In some situations, the research data can be collected without the use of client- or server-side scripting. Click-stream data, for example, can often be gleaned through the use of standard network management software, or through *network sniffers* that can be configured to monitor Internet requests and page downloads. In this case, the experimental treatment can involve pages other than those created specifically for the research study, and, again, standard browser software can be used for the experiment. The problem here can be in the precision or in the format of the data, as the software was not designed for this purpose. Pages containing multiple frames, for example, may be logged as individual (frame) downloads in some circumstances and as a single-page download in others. Client requests that are satisfied through the local cache may not be logged at all.

A problem with all of the data collection methodologies discussed thus far is that they suffer from a lack of experimental control. This lack of control comes from the fact that the instrument with which the experimental subject is interacting (a standard Web browser such as IE) was not designed to be used as a research tool.

Consider the situation in which we wish to study WWW use behavior through analyzing click-stream data. There are ways of gathering data on page requests or page downloads, as noted above. However, there is no means, short of direct visual observation, of recording how a particular page was requested. The page request could have come in the form of a click on a hyperlink, but the request could just as likely have been generated automatically through a dynamic action on the page (e.g., *meta refresh*), or through the *Back* or *Forward* buttons in the browser interface. Normal click-stream data will not distinguish between these circumstances, so the precise behavior or intentions of the experimental subject cannot be determined.

Another problem has to do with the occurrence of multiple windows. Many Web sites open hyperlinks in new browser windows. The problem here is that the data collected cannot reflect which of the open windows is active when actions occur, or even that there are multiple windows in use. Again, the data cannot capture or misrepresents the behavior in question; true *streams* cannot be traced.

A CLIENT-SIDE SOLUTION

As noted earlier, the methodological problems, for the most part, stem from a lack of experimental control. Logic and research experience suggest that, for maximum experimental control, the experimental manipulations (treatments) and the data collection mechanisms should be as close to the experimental subject as possible. That is, they should be embedded in the browser. This leads to the development of a custom IE-lookalike browser for use in Web-based experiments. The creation of such a software application is feasible with currently available programming tools and software techniques. The numerous benefits of this approach certainly outweigh the software development costs. The benefits are greatest when research designs are complex and when precision is of prime importance. This particular methodology has been employed in several research studies, including Norberg (2003) and Chiang et al. (2004).

With custom browser software, there is no need to depend on scripts or applets in experiment-specific Web pages to administer experimental treatments or to record user actions. Consequently, there is no need to restrict the experimental domain to a limited set of custom Web pages. With this approach, the experimental domain can include the entire Web. The custom software can be built with the ability to precisely record user activity and to preempt or modify actions that could be harmful or inappropriate in the experimental context. Experimental control and experimental manipulation can be integrated into the browser.

The software that we know as Internet Explorer is essentially a software interface surrounding a set of dynamic link libraries (DLLs) that provide the requisite Internet processing functionality. Microsoft, in its *Visual Studio* suite of software development products, provides a software object called the *WebBrowser Control* (see Microsoft Corporation, 2004a). This control can be used in Visual Basic (VB) or in C++ applications to add Web browsing functionality to software applications. The *WebBrowser Control* can be thought of as an object wrapper around the Internet-processing DLLs noted above. The *WebBrowser* object works with the standard event-based model of Windows computing.

With the *WebBrowser* control, event handlers are provided for all of the major occurrences in an Internet session, such as *request to navigate to a page*, *page download complete*, or *request for a new window*. Key data such as URL, Target Frame, and Page Title are available with the events. In some cases, actions can be preempted through a *Cancel* argument in the event handler. One important example of this is the *BeforeNavigate* event handler. This routine is triggered after a navigation has been requested by the client, but before the request is fulfilled. This allows the custom software to inspect and evaluate the situation and to possibly modify or cancel the request before it is allowed to proceed.

Properties and methods of the *WebBrowser* object can be used to dynamically emulate all of the features of the IE interface, such as the status bar, the browser window caption, and the standard buttons (Back, Forward, Stop,