An Environment for Creating Interactive Statistical Documents

Samuel E. Buttrey* Deborah Nolan†
buttrey@nps.navy.mil nolan@stat.berkeley.edu
Duncan Temple Lang‡
duncan@research.bell-labs.com

Abstract

The spectacular growth and acceptance of the Web has made it a very attractive medium for interactive documents. Web-based reporting in industry, "live" documents in research, and interactive worksheets in education material are in many ways ideal uses of the Web. These types of documents frequently display dynamic, statistical output both in the form of text and plots. Unfortunately, much of the effort in creating these types of documents has focussed on re-inventing existing statistical software, and often with inferior results. The reason is that systems such as S and SAS cannot be integrated into the reader's browser.

A better approach is to allow the author to create a document using the common authoring tools (e.g. LaTeX, MS Word or HTML editors) and to conveniently insert dynamic and interactive components from other languages. The author focuses on the presentation and display of these components, including the usual multi-media elements such as text, images and sounds. She uses HTML form elements and Java components to provide interactive controls with which the reader can manipulate the contents of the document. And finally she performs statistical computations and renders visual displays using the statistical software that is embedded within the reader's browser.

In this presentation, we describe how we have created an environment for interactive statistical documents. It allows the author to use HTML, JavaScript and R to create the content and the interactivity. The reader accesses the interactive and dynamic functionality of the document via a plug-in for Netscape that embeds R within it. The different languages are all reasonably standard tools and each is used for the purposes for which it was designed. This makes it a reasonably straightforward environment in which to quickly and simply create interfaces for various different applications and audiences.

1 Introduction

The Web has provided a significant opportunity for statisticians to present analyses in new and exciting ways to a much wider audience. Providing dynamic and interactive documents that involve users and encourage them to explore methodology, analyses and inferences within an environment with which they are familiar and comfortable is now feasible and increasingly expected. Researchers, business people and others confronted with rapidly-growing data sets need technology that will permit them to investigate their data without learning specialized statistical tools.

Many people have recognized this opportunity within the past several years, and have embarked on significant projects to create Web-based reporting environments for industrial use, interactive research papers that allow readers to explore the characteristic of methodology, and in teaching statistical concepts to students who are already familiar with interactive, multi-media environments. Many of these projects have made extensive use of Java because it is portable across different operating systems and runs both as applets in browsers and as stand-alone applications. However, in the domain of statistics, the promised advantages of Java are contrasted with the inability to easily use existing libraries of statistical software. Hence, many of these projects have had to re-develop all of the basic statistical software and have not benefited from the years of use that stable tools have achieved. As a result, valuable resources are consumed and the results have been, generally, inferior to existing software.

*Department of Operations Research, Naval Postgraduate School, Monterey, CA
†Department of Statistics, University of California at Berkeley, Berkeley, CA
‡Statistics and Data Mining Research, Bell Labs, Murray Hill, NJ

This inability to use existing statistical software environments in Web browsers (and in other, more general applications) has motivated us to provide the functionality of the statistical environment R[**?**] within the Netscape browser. The Omegahat Project (`www.omegahat.org`) has developed a plug-in for Netscape which allows authors to easily integrate multi-media content and statistical computations and visualization. In this environment, an author creates a document using HTML to format and display the different elements. These elements consist of the usual text, images and sound. The author can employ more sophisticated components supported by the browser, such as HTML forms, Java applets, movies and Flash plug-in displays. She can also use JavaScript to respond to user interaction in customized ways. Our extension to this environment is to make the statistical environment R accessible from within JavaScript code and also to allow R graphics devices to be embedded directly within the contents of the document.

Using Netscape's LiveConnect facility, JavaScript code can call R functions as if they were regular JavaScript functions. These calls are evaluated in a regular R session running within Netscape and local to that page. The session is created like any other plug-in, using the <**EMBED**> tag and the MIME type `app/s-xinterpreter`. Data structures and values are converted between R and JavaScript automatically, with primitives and objects in one system mapping to primitives and objects in the other, and JavaScript arrays mapping to R vectors and *vice versa*. In addition to calling R functions directly, one can also evaluate R commands specified as strings.

One can program the components of the document in any of several different languages, choosing whichever is most appropriate for the particular task. For those more familiar with R than JavaScript, it may be more convenient to develop the interactivity within R, treating JavaScript as a library of high-level classes and functions rather than a programming language. The symmetry of the R-JavaScript interface makes such a view possible, as R code can access and manipulate JavaScript objects and their methods and fields.

The aim of this authoring environment is simplicity and flexibility. The integration of the different languages (HTML, JavaScript, R, Java, and other plug-ins) is a luxury that allows one to program where it is most convenient. We can avoid using a language that is ill-suited for a certain task and makes the programming unintuitive and difficult to maintain. The complexity of programming in numerous languages is hopefully eased by the fact that each of them is well-documented and somewhat standard in its own domain. It is this use of "off-the-shelf" tools and their direct integration that makes this approach more powerful and potentially acceptable than other application- and domain-specific approaches. Additionally, the modularity and separation of the languages allows different authors to develop different elements. For example, an R expert may create the underlying functionality independently of the Netscape and JavaScript and the document author can integrate these simply by calling the functions.

2 Examples

In this section, we will briefly discuss two examples of interactive documents. One is a pedagogical example and shows how we can quickly use the interactivity to allow the student to explore statistical concepts in a fashion more interesting than simply reading text books or performing simulations programmatically. The second example is an illustration of Web-based reporting and how interactivity coupled with powerful statistical facilities can provide information that is easy to access at different levels.

2.1 Central Limit Theorem

In this example, we show how simple it can be to put together an interactive document to demonstrate the Central Limit Theorem. A number of authors have produced Java applets showing the effects of the CLT; in our case we use the existing random number generation and plotting tools within R. We make no claims about the pedagogical advantage of the demonstration presented here over these applets, but we do think that it has advantages over most applets with respect to the user interface, extensibility, and time investment in developing it.

In the document, we lay out the areas of user input in HTML, handle events through JavaScript, and generate random numbers and figures in R. The student clicks on a radio button to choose a probability distribution (see figure 2.1). The selection causes the evaluation of a JavaScript expression which uses the R graphics device embedded in the HTML page to display a plot of the distribution.

The student selects a statistic from a pull-down menu and specifies the sample size. To generate a sample, the reader clicks the **Sample** button. A JavaScript expression handles this by retrieving those inputs and passing them as

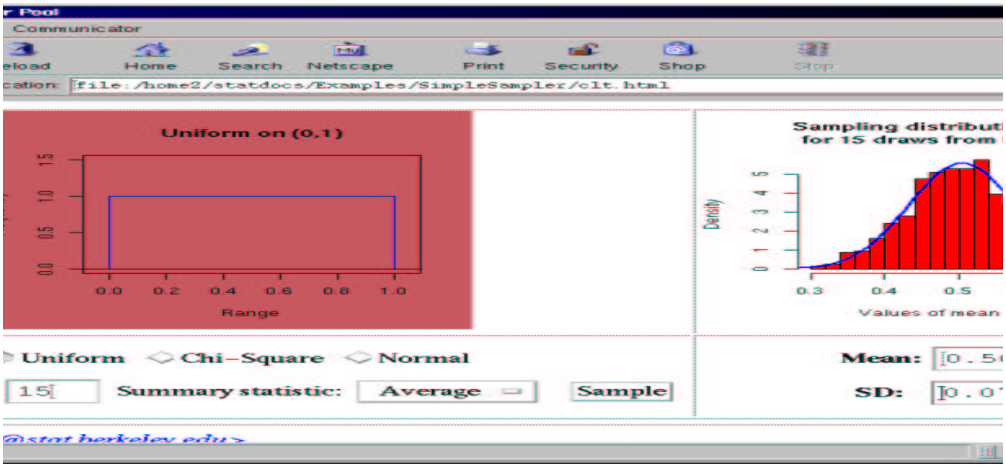

Figure 1: The Interactive Central Limit Theorem Demonstration

values in a call to an R function which generates 1000 samples and computes the corresponding values of the chosen statistic. R then plots the simulated sampling distribution, and returns its mean and SD. The JavaScript command then displays these numbers in the corresponding text fields under the plot.

The formatting is done by tables in HTML, using the <**EMBED**> tag, and forms for the radio buttons and text fields. We use R's well-tested and widely used code for doing things such as generating the pseudo-random numbers and placing tick marks on axes. This allows us to easily customize the graphs by using R's graphical controls. Additionally, we can easily extend our document to include any probability distribution (e.g. Cauchy, Weibull) or any summary statistic (e.g. trimmed mean, quantile). We can also add controls to, for example, parameterize the distributions or specify arguments to the statistic. Further, it is simple to change the HTML that controls the layout and inputs and to include this document in a larger document. In contrast an applet will need to have been designed to support such extensibility and its elements cannot be placed independently. Finally, we can "animate" the demonstration, and build the sampling distribution for the reader by using JavaScript timers.

2.2 Web-based Reporting

A wafer fabrication plant produces silicon wafers onto which chips are etched. The wafers are produced in "lots" of about 50 wafers each, and there are about 150 lots produced each day. Each wafer holds about five thousand chips. Each chip is tested and an error code (including a "good" code, indicating an errorless chip) is assigned to each chip. Engineers in charge of the production process are interested in the proportion of good chips by wafer and by lot, and in the spatial patterns of errors by both wafer and lot. Information on the rates and locations of failures can be used by engineers to understand and correct difficulties in the production process. To this end, we generate interactive reports that these engineers can access daily via their Web browser.

The basic report is simple. It presents a histogram displaying the distribution of the yield across all lots and wafers and a table listing the minimum, maximum and average yield for each lot. This allows the engineer to quickly determine if all is well and if certain lots are very different from others. If no anomalies appear, the engineer may stop reading the report. However, we provide a way for the engineer to examine the data in more detail.

Figure 2.2 shows a display that the engineer can use to query the characteristics of each lot. A pull-down or choice menu allows her to select the lot of interest. Since the data for each lot are stored in R, we use R to compute the number of wafers in the lot and a summary of its yield each time a lot is selected. We also compute the frequency table

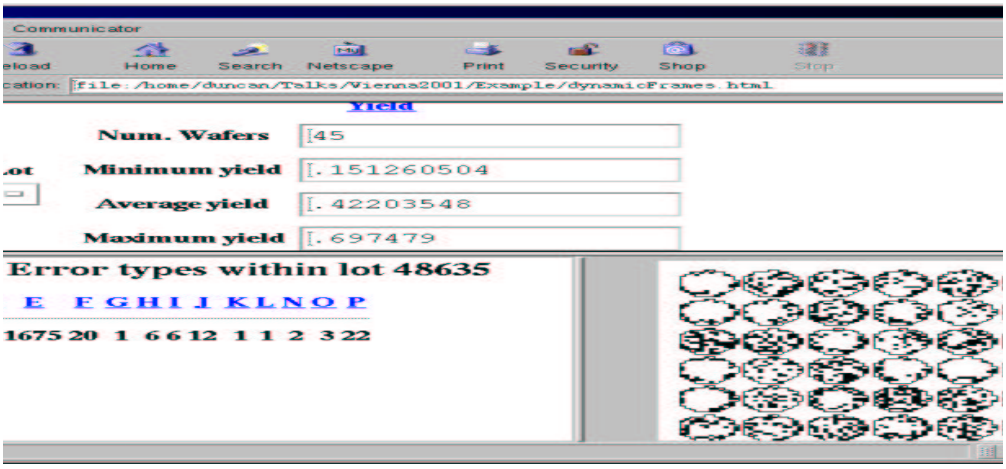

Figure 2: An Interactive Wafer Report

of the different error categories for the lot, and display it both textually and graphically as a bar plot and, alternatively, a wafer plot of the spatial distribution of each error type.

Now we turn our attention to how this document is constructed. We use HTML frames to partition the display into the three different regions: the summary controls and display, the error type table and the bar or wafer plot. The summary controls are displayed using an HTML form and <**input**>. Rather than enumerating the list of lots in the document, we dynamically fetch it from R and have JavaScript add each element to the pull-down menu. This makes the document independent of each each day's data.

We attach an action to the <**SELECT**> menu in the summary controls which is invoked each time the reader selects a lot. This is specified as a JavaScript expression and is responsible for fetching values from R and then adding them to the display. For example, to update the summary information, the JavaScript function *showLotRange()* is defined as

```
function showLotRange(lot) {
 var form = document.forms[1]

 form.numWafers.value = R.call("getNumWafers", lot);
 var vals = R.call("getLotInfo", lot);
 form.min.value = vals[0];
 form.avg.value = vals[1];
 form.max.value = vals[2];
}
```

There are two calls to R, one to get the number of wafers in the lot and one to get its summary statistics. The R-JavaScript bridge automatically converts the R vector to an array of numbers in JavaScript. We can then immediately display the values in the different text fields in the form by setting the **value** field in each of these elements.

Displaying the error table is done within JavaScript by calling an R function that returns a named list of error type-value pairs. We iterate over this table in JavaScript and write the HTML version of the table directly into the document in the lower left frame.

The graphical display in the lower right frame uses a special form of an R graphics device. We specify a region within the HTML document and insert the graphics device using the HTML tag <**EMBED**>.

```
<EMBED type="app/x-sgraphics" name="plot" WIDTH=300 HEIGHT=300>
```

We can then use this as a regular R interpreter, but with additional JavaScript methods for creating plots. In our case, to create the bar plot we invoke a slight modification of the standard `barplot()` function in R which we have named `lotBarplot()`. The call `R.plot.call("lotBarplot", lotId)` causes the plot to be displayed directly in this graphics device. A analogous action produces the wafer plot visible in the figure.

As a final control, the engineer can click on any of the error categories in the table. This replaces the bar plot with a plot of the individual wafers within the lot, identifying the locations at which that error type was observed. Again, this uses existing R code and involves merely catching the user's click and calling an R function from JavaScript giving it the lot and error type. This easy extensibility is one feature that makes this approach attractive.

3 Other Approaches

Sawitzki [**?**] has demonstrated a style of interactive research report using the more-specialized statistical environment Voyager. TILE (`www.stat.berkeley.edu/~tile`), SticiGui (`www.stat.berkeley.edu/~stark/SticiGui`) and others have used Java to provide the interactivity, but have had to re-develop the necessary statistical tools for education. Matlab, R and other scientific computing languages have added support for building graphical user interfaces. These shift the reader into these environments rather than the familiar browser. The use of Netscape as the containing or host application is a new approach and is a natural choice for less-technical audiences who are unfamiliar with R.

The notion of including output from one application in that of another is common in the Microsoft and Apple environments, and is emerging in Unix via the KDE and Gnome tools. This produces documents that are often in a proprietary format and often not Web-friendly.

We are also experimenting with combining output from R into interactive statistical documents created using XML templates.

4 Future Work

Immediate work on the plug-in will provide support for security considerations, and separate, individual interpreters for different pages. The *StatDocs* project (`www.stat.berkeley.edu/~statdocs`) is an effort to create libraries of XML/XSL templates, R functions and JavaScript functions for rapidly building interactive documents. We will also provide many example documents in the different areas of research papers, web reporting and pedagogical material.