# FastRWeb: Fast Interactive Web Framework for Data Mining Using R

Simon Urbanek [1]

[1] AT&T Labs - Statistics Research, Florham Park, NJ, USA.
E-mail: urbanek@research.att.com

**Keywords:** data analysis, web based methods, R, data mining, interactive graphics

## Abstract

R is widely used and accepted as a very versatile tool for statistical computing and data analysis. It provides a plethora of cutting edge methods, tools and algorithms. However, typical data analytic work entails the use of command line that requires a good knowlegde of the language. On the other hand the World Wide Web (the Web) infrastructure represents a technology for wide deployment and high accessiblity. With current browsers and the dynamic technologies such as AJAX it is now posssible to create highly interactive content. Our basic goal is to fully leverage R and yet to allow users to interact with the system without the need to resort to the R language. Other interpreted languages are routinely used on the Web. R has seen a slower adoption in this area mainly due to the lack of high-level web support and its high start-up costs.

We propose a framework that addresses both issues and allows very fast responses. It also provides building blocks not only for reports, plots and analyses, but also fully interactive graphics. In addition it is highly modular, allowing a maintainable creation of complex user interfaces for reporting, monitoring and data analysis.

In this talk we will describe the various parts of the system ranging from the fast-response infrastructre, AJAX tools for on-demand data loading to R facilities for intuitive creation of web objects, plots and interactive graphics. We will illustrate the use of the framework on several examples, including our implementation of a real-world mining tool used in practice for exploratory data analysis and data mining in very large databases. The use of Web-based methods allows us to use one system to target both users without statistical knowledge and domain experts.

## 1 Introduction

The Internet has profoundly changed the way we work nowadays. It has introduced a common infrastructure not only for content delivery, but increasingly also for fully interactive work. Today's browsers supporting dynamic content and rich user interaction are ubiquitous. Precisely the availability makes it possible for many users around the world to use a common application from any computer which is the true strength of the Web.

On the other hand data analysis, monitoring and presentation are usually performed using very specific tools that are difficult to share and deploy. At the same time, especially complex analyses are often performed on remote machines with restricted interaction possibilities. Although attempts have been made to provide access to such tools through Web technologies, it is mostly limited to static snapshots and pre-generated content. In this paper we want to discuss all parts involved in using Web technology to leverage the full potential of an interactive statistical environment provided by R for data analysis and visualization. We will also introduce a highly responsive infrastructure FastRWeb that allows users to create dynamic web content very easily. The presented tools provide a platform that can be used in a wide range of settings — ranging from quick experiments, ad-hoc analyses, educational webpages to complex interactive data analytic systems.

In the first section we will discuss the fundamental parts of performing data analysis through Web technologies, in the second section we will describe the FastRWeb system implementation and highlight its benefits. In third section we will illustrate the use of the system on several examples and conclude with an outlook for future research and a summary.

## 2 Interactive Web Content

The aim of the framework is to hide all technical complexities from the user, resulting in a flat learning curve. All the content creator has to provide is a function for the content to be displayed, such as a plot, summary or description. The system will perform all necessary steps to create and deliver the content on demand.

The approach is very similar to that of using scripting languages such as perl or PHP to generate content. The major difference here is that we will be using R as the scripting language and provide seamless handling of complex objects specific to the statistical analysis such as graphics, images and interactive plots. Further the use of a functional language such as R opens entirely new possibilities of expression that are not available in commonly used scripting languages. Finally, R brings a large selection of packages to the table, providing cutting-edge technologies to the analysts.

From developer's perspective the concept is very simple: each part of the content is specified by an R function. The function can take arguments that can be arbitrary or based on the state of the session. The result of the function represents the content and can range from raw HTML code to interactive plots. A simple example of such a script that produces a plot looks like this:

```
# file: kmeans.png.R
run <- function(clusters=3, ...) {
  cl <- kmeans(x, clusters)
  p <- WebPlot(600, 400)
  plot(x, col = cl$cluster, pch=19)
  points(cl$centers, col = 1:clusters, pch = 8, cex=2)
  dev.off()
  p
}
```

The `run` function represents the entry into the script. It will be supplied with parameters from the actual web request. The above function performs a k-means cluster analysis on the data `x`, plots the data colored according to the cluster and adds symbols for the cluster centers. The value returned by the function is the generated plot which will be shown in the browser. `WebPlot` is conceptually equivalent to opening a bitmap graphics device such as for example `Cairo` and the first two parameters specify the width and height of the plot in pixels. The URL request for this page could look somewhat like `http://myserver/R/kmeans.png?clusters=4`. The `cluster` parameter is optional and will default to 3 if not specified.

The above example assumes that the data `x` is already preloaded in the system (which we will discuss later), but it could be loaded from a file or database in the function. For example add `x <- faithful` at the top of the function for a stand-alone example.

The interactivity of the content is created by a combination of parametrized scripts such as the one above and user interface elements. Those can be hyperlinks, queries or even AJAX requests as we will show. Additional interaction such as fully self-contained interactive graphics can be created using tools in R, but it exceeds the scope of this paper and will be handled separately.

Nonetheless we shall discuss the parts involved in the way that leads form a script like the one above to the browser where the result is displayed. The whole picture is shown in Figure 1:

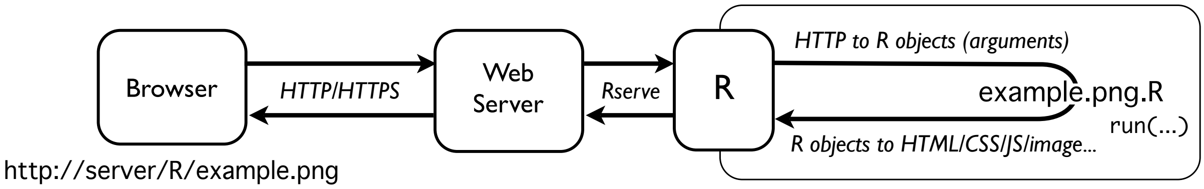
Browser
HTTP/HTTPS
Web Server
Rserve
R
HTTP to R objects (arguments)
example.png.R
run(...)
R objects to HTML/CSS/JS/image...
http://server/R/example.png

Figure 1: Processing a Web request.

The browser provides a URL which identifies the script to run and any additional parameters. This URL is processed by the web server and passed to R. Tools in R have to interpret those parameters, create corresponding R objects, parse the R script and call the `run` function. The result of the function has to be converted to a form that the browser understands, e.g., HTML document has to be generated, possibly also including any CSS or JavaScript parts that are necessary for the interaction. This result is then delivered to the client and displayed in the browser. The three major components of the process are:

- **communication** - this part is responsible for connecting web server and R in the most efficient way such that immediate response is possible
- **request input processing** - this part makes sure that input data are processed into R objects that can be easily used
- **output generation** - this part creates content that can be rendered in a web browser from R results. This includes generation of images, HTML documents, CSS or even JavaScript.

Tools on the R side are responsible for the creation for R objects from requests as well as generating HTML code from R objects. The input processing task is performed automatically by the FastRWeb infrastructure. For the task of output generation several tools already exist at various levels of abstraction such as R2HTML, brew or WebGraphics.

## 3 Implementation

The major impediment of previous attempts to use R on the Web such as Rweb by Banfield (1999) or CGIwithR by Firth (2003) is the slow response time caused by the initialization delay of R and thus the communication part. This problem is further attenuated when large data have to be loaded for the processing. This makes it unfeasible to use such systems to build highly interactive and dynamic content which has to be delivered within milliseconds.

FastRWeb addresses this issue by using Rserve back-end for R invocation, see Urbanek (2003), which has many benefits including instantaneous response and preloading of code and data. Rserve circumvents the initialization problem by fully initializing R first and then listening for incoming requests. The initialization can take substantial amount of time without impacting the response time. This can include the loading of packages or shared data.

In addition, parallel requests are supported as well, which also reduces the response time where multiple content parts are displayed on a single page such that they can be computed in parallel. Conveniently, all such request share the initial data which saves memory since only modification will result in separate memory usage.[1]

Another benefit of using Rserve is the possibility to separate the web server and the R server. Especially in large applications this allows for a pool of R servers to serve a common web server, off-loading the computation to separate machines. Nonetheless, Rserve can also be used locally on the same machine as the web server, communicating via local sockets for extremely fast connection.

FastRWeb can be used with an arbitrary web server since it uses established and widely-available Common Gateway Interface (CGI, Internet Engineering Task Force (2004)) to communicate between the web server and Rserve. This CGI/R layer is very thin, consisting of a single, small executable that connects R with the web server. Unlike heavy-weight interpreters, this approach of a light-weight layer reduces the delay and competes very favorably with more elaborate web server embedding, which is not as general. The communication layer parses incoming URL (both the path as well as the query string) and generates corresponding R code to generate arguments for the `run` function as well as the code to parse and evaluate the R script.

For example a default configuration of a web server with `cgi-bin` reserved for the CGI interface and `Rcgi` executable for the CGI/R layer, the URL `http://server/cgi-bin/Rcgi/foo/bar.html?a=10` would be interpreted by the `Rcgi` layer to parse the script `foo/bar.R` (the suffix `html` is replaced by `R`) and then evaluate `run(a=10)` in R. The result is then garnished with corresponding HTTP headers and returned back. The script is looked up in the file system in a way analogous to a web server looking up html files. This makes it very easy to create highly modular applications, simplifies debugging and allows fast prototyping since any changes are effective immediately.

Intermediate results such as plots can be shared on disk or transported through the CGI layer. FastRWeb has a built-in cleanup system that ensures removal of temporary files after they have been

[1] Features described here apply to the unix version of Rserve. We do not recommend the use of Windows-based servers due to severe limitations of the operating system that does not allow parallel processing and shared instances of R.

delivered. Although it would be feasible to build a pure memory-based graphics delivery, it would require a modification of R graphics devices. Modern operating system will still cache such files in memory, so the benefit of the more elaborate approach is not evident.

In addition to the communication and data input aspect, FastRWeb also provides simple tools for creating content. The most basic facility is to create HTML output sequentially, in the same fashion as textual output is created in R:

```
run <- function(n=20, ...) {
  out("<h2>Linear Model</h2>")  # plain HTML
  d <- data.frame(x = rnorm(n))
  d$y <- d$x + rnorm(n) / 5
  m <- lm(y ~ x, d)
  oprint(m)  # verbatim output
  p <- WebPlot(600, 400)
  plot(d$x, d$y ,pch=19, col="#ff000080")
  abline(m)
  out(p)     # objects/graphics
  otable(d)  # table
  done()     # create the result
}
```

The most simple function `out` simply appends the arguments in textual form as-is to the content. By default the content is assumed to be a HTML document. Special objects such as web plots have corresponding methods that create the HTML for you. Since R output is usually not in HTML form, `oprint` appends the literal output of `print` in a form that will be rendered correctly in the browser. Other convenience functions exist such as `otable` which facilitates the creation of tabular output. All output functions simply append to the HTML document created so far. The full document is obtained using the `done` function in a form that is proper for the delivery engine.

Alternatively FastRWeb makes it possible for advanced R scripts to manually construct arbitrary HTTP responses via the `donehttp` function. This includes full control over header fields such as `Content-Type`, `Set-Cookies` as well as the body. Advanced applications such as session management and access control can be implemented this way. Such advanced topics exceed the scope of this paper but are documented as examples in the FastRWeb package.

The technical implementation of FastRWeb is what makes it possible to create a highly interactive system. We have deployed FastRWeb successfully in a wide variety of applications ranging from simple monitoring to full-scale data mining tools. Although it is not possible to describe such systems in a paper, in the next section we want to illustrate the underlying concepts on a few practical examples.

## 4 Examples

We have seen in the previous section several examples of creating content in R. However, the real benefit of FastRWeb is the ability to provide very fast, immediate response. This allows us to fully leverage the aspect of interactivity. One of the key requirements for interactive work are queries (see Unwin (1999)). FastRWeb provides a global query object that can be use to display arbitrary queries. In our data mining tool we show a list of recommended TV programs of which each can be queries by moving the mouse over the program name. This example illustrates the process:

```
# proglist.R
run <- function(...) {
  # program list "p" is created
  # ...
  # then displayed
  otable(alink(p$title,
               onClick=arequest("add_program", "programs", id=p$id),
               onMouseOver=queryReq("pinfo", id=p$id),
               onMouseOut=queryOff()))
  done()
}
```

```
# pinfo.R
run <- function(id, ...) {
  p = prog[match(id, p$id),]
  if (dim(p)[1] == 0) return("Program not found")
  # optional CSS for aesthetics: yellow background, black border, 5 pixel margin
  out("<div style='background:#ffffe0; border: 1px solid #000; padding: 5px'>")
  # show program title (in bold) and description
  out("<b>", p$title, "</b><br>", p$descr, "<p>")
  # show all other fields of the program table if populated
  for (i in 4:length(p))
    if (!is.na(p[1,i]))
      out(paste("<br>",names(p)[i],": ",p[1,i]))
  out("</div>")
  done()
}
```

The query is activated by adding `onMouseOver` and `onMouseOut` events to the link of the program. The query content is produced by another R script which is parametrized by the program identifier `id`. Both functions `arequest` and `queryReq` create corresponding HTML to issue an asynchronous (AJAX) request to fulfill the need. The former replaces part of the content on a page (element named "`programs`"), the latter creates and displays a query. Since FastRWeb is fast enough to respond within milliseconds, the query can be offloaded to another script "`pinfo.R`". That script looks up the program details and returns them in a yellow box with a border. The code shown above illustrates the use of HTML output directly, but it is possible to achieve the same result using higher-level abstraction. The retrieval of data, here with a pre-loaded `prog` variable, can be equally easily (and quickly) performed by a database lookup.

The concept of loading parts of the page on demand is crucial for interactive applications. It allows us to provide a separate script for a menu, sub-items, graphics and conceptually separate results. FastRWeb also has the ability to inject JavaScript programs into an existing page, making it easy to load and unload parts of the page as necessary. This ability to replace parts of the document on demand gives us a basis for another important interactivity concept: change of parameters.

In our first example in section 2 we have already seen a parametrized output which depends on the predefined number of clusters passed as a parameter `clusters`. All we need now is to add user control for it:

```
# clusters.R
run <- function(...) {
  out(img(id='kmeansPlot', src='kmeans.png'))
  out("Set the number of clusters: ")
  out(alink(1:10, onClick=
    setAttr("kmeansPlot", src=uri("kmeans.png", clusters=1:10))))
  done()
}
```

After referencing the generated image we add a series of links with numbers 1 to 10 which are hyperlinked to set the source of th image to "`kmeans.png`" plus `clusters` argument reflecting the number. Analogously it would be possible to create a text field and set the `clusters` parameter to the number entered there or use a drop-down list. In the latter approach we can use the fact that HTML form fields are passed as arguments and thus require no special handling.

The above example used `setAttr` to modify the `src` attribute of the `img`. Instead of modifying the image directly, a more general approach is to define a division/section using the `div` tag and replace its entire content on a click or other interaction. This allows not only flipping images but also dynamic change of the whole content in a division. That is the most common way to modularize the web document.

## 5 Future Work

FastRWeb opens a door for any content to be delivered on the web. Although it provides basic tools for content creation, the possibilities for web content are endless. Further packages can expand on the idea and provide tools that further simplify the content creation and add new, more complex content such fully as interactive graphics or automatically generated scripts. WebGraphics package is a first step in that direction of future development. The research opportunities are not restricted to implementations – new frontiers in the combination of functional languages and describing interactive content can be forged.

Rserve is ideal for dynamic web content generation using R due to its ability to pre-load code and data. However, it currently doesn't support modification of the pre-loaded data. Updates of the data essentially require a re-start of Rserve. It would be helpful to add an update facility which allows modification of the preloaded data by executing arbitrary scripts in the main R instance. This is technically possible but remains to be implemented[2].

## 6 Conclusion

The FastRWeb system makes it possible to deploy and develop web based content very quickly. Many applications can be built using this infrastructure ranging from webpages to complement teaching and education, ad-hoc result presentation, monitoring, alerting and data mining in large databases. The flat learning curve for R script developers makes it very easy to add new content and tweak existing parts. Automatic parsing of request parameters to function arguments offloads the administrative burden. Support for asynchronous requests (AJAX) allows an easy modularization of the content, simplifying debugging and maintenance.

The use of existing infrastructure and well-defined interfaces such as CGI makes it very easy to deploy FastRWeb on standard web servers. The thin layer between the webserver and R using CGI and Rserve allows separation of computing and web delivery while maintaining vert fast response time. The ability of Rserve to pre-load data and code (including packages such as database drivers) makes it possible to build results very quickly and interactively. Basic output facilities such as textual output, summaries, tables and plots are seamlessly integrated in the output. Additional modules for advanced web-interaction such as authentication and access control are available.

## References

Banfield, J. (1999). *Web-based Statistical Analysis*, Statistical Software, **4**, 1, 1–15

Firth, D. (2003) *CGIwithR: Facilities for processing web forms using R*, Statistical Software, **8**, 10, 1–8

Internet Engineering Task Force (2004) *The Common Gateway Interface (CGI) Version 1.1*, RFC 3875, The Internet Society

Unwin, A. (1999) *Requirements for Interactive Graphics Software for Exploratory Data Analysis*, Computational Statistics, 14, 7–22

Urbanek, S. (2003) *A Fast Way to Provide R Functionality to Applications*, Proceedings of the 3rd Int. Workshop on Distributed Statistical Computing (DSC-2003), 1–11

brew package, Horner, J., `http://www.rforge.net/brew/`

FastRWeb package, Urbanek, S., URL: `http://www.rforge.net/FastRWeb/`

R2HTML package, Lecoutre, E., URL: `http://cran.r-project.org/web/packages/R2HTML/`

Rserve package, Urbanek, S., URL: `http://www.rforge.net/Rserve/`

WebGraphics package, Urbanek, S., URL: `http://www.rforge.net/WebGraphics/`

[2] Note: this has been implemented as *control commands* in Rserve version 0.6 after this paper was published.