Web Development with R

UCLA Statistics Seminar

Jeroen Ooms
jeroenooms@gmail.com

November 2009, Los Angeles

What is a Web Application

Some examples:

- yeroon.net/ggplot2 [video]
- yeroon.net/lme4 [video]
- Pubertyplot [video]
- Stockplot [video]
- IRT tool [video]

What is a Web Application

This presentation is about:

- Making Web Interfaces for Simple R applications.
- R runs on the server.
- User only needs a standard webbrowser.
- Programming in R and HTML/Javascript (No Java!).
- AJAX interfaces.

It is not about:

- Applications that require advanced queueing systems for running multiple simultaneous R jobs.
- Distributed Computing.
- Java.

Why Web Applications?

Convenient:

- Making new tools available to a wide audience.
- User only needs a standard webbrowser.
- Cross-platform.

Server-based by design:

- Efficient use of resources.
- Easier to maintain.
- Integration with existing services/databases/etc.

Because you have to:

- More and more services only available from the browser.
- Many people can't or dont want to install software.

Hello World!

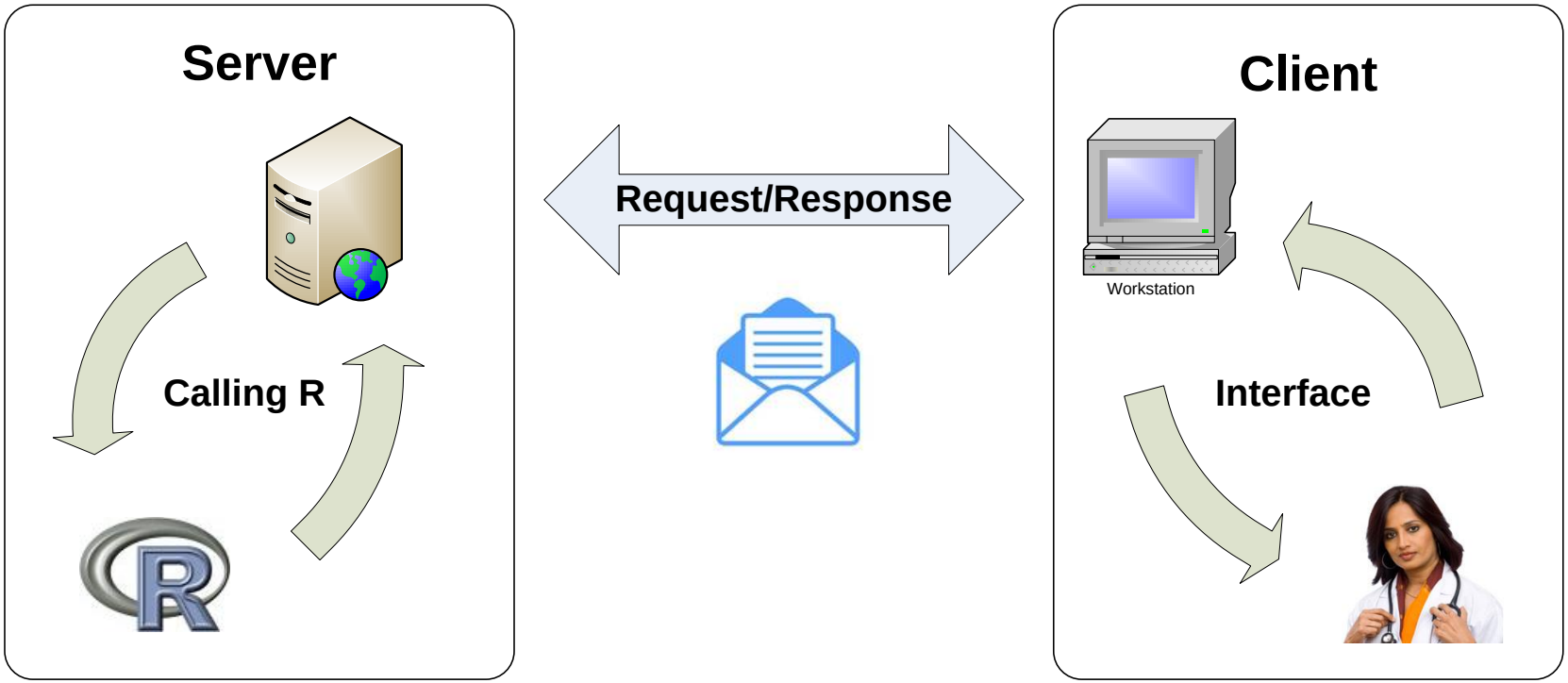
RApache

- RApache is a module for the Apache web server.
- It allows you to embed R scripts in a webpage.
- It is one of the ways of connecting the HTTPD to R.

Demo:

- Brew scripting.
- GET/POST.
- Try Catch.

Web application Setup



ggplot2 example

1. Interface:
   - Interactive webpage with menu, buttons, layer list, etc, that enables the user to specify a plot.
2. Request:
   - A configuration of a plot to draw (settings, mappings, layers)
3. Server:
   - Decodes the configuration, and draws the requested figure.
4. Response:
   - The plot figure.
5. Interface:
   - Updates figure within the page, and makes appropriate changes in GUI.

The Message Specification

Where the magic happens:

- How to get objects from the client's browser into R on the server?
- HTTP basically only takes strings for arguments.

Protocol Design:

- Maintain a request object in the browser (client).
- Define a way to encode your javascript object into a string.
- Use this string as argument to an HTTP/AJAX request.
- Decode the string on the server back to an object.
- Idem for the response message.

Message Specification: XML and JSON.

- For advanced web applications use a standard data format to communicate with the server.
- 2 obvious candidates: XML and JSON.
- You can use CRAN packages XML and rjson for parsing.
- XML is most widely used, probably preferable if you want to integrate your application in other software (consider SOAP).
- JSON is lighter and supports array's. Preferred for large datasets.

Example of XML Syntax

```
<myModel>
    <family>Gaussian</family>
    <deviance>3569.23</deviance>
    <coefficients>
       <coef>
           <name>Intercept</name>
           <value>5.69</value>
       </coef>
       <coef>
           <name>Age</name>
           <value>0.36</value>
       </coef>
       <coef>
           <name>Gender</name>
           <value>2.54</value>
       </coef>
    </coefficients>
</myModel>
```

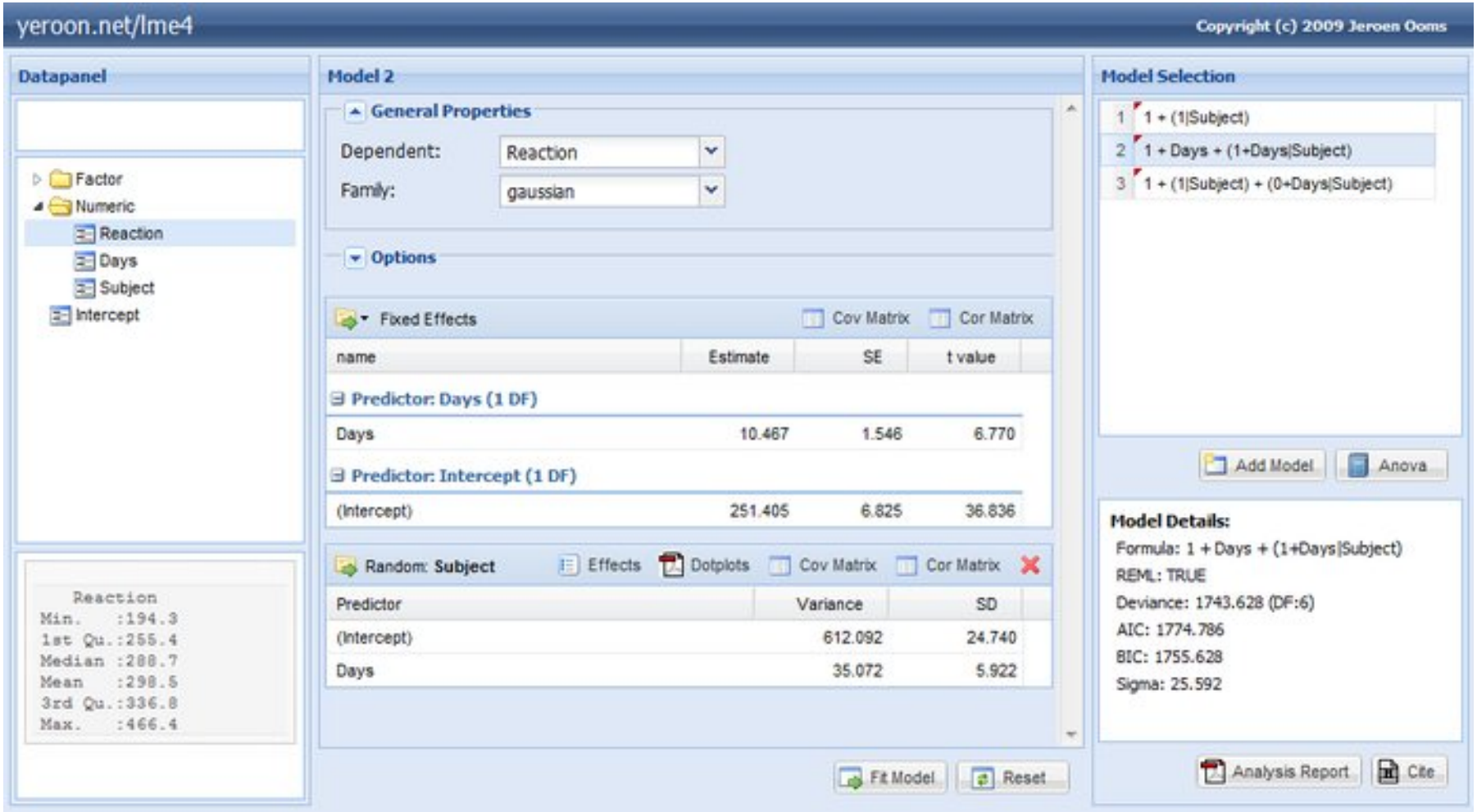
Example of JSON Syntax

```
{"myModel": {
    "family": "Gaussian",
    "deviance": 3569.23,
    "coefficients":
        [ {"Intercept": 5.69}, {"Age": 0.36}, {"Gender": 2.54} ]
    }
}
```

Or for example a dataframe:

```
{"myData": {
    "Age":[9,8,12,6,7,8,9,8,10,11,9,6,8],
    "Gender":["M","F","F","M","F","M","F","F","M","F","M","F","F"],
    "Treatment":[1,0,0,1,1,1,0,0,0,1,1,1,0]
    }
}
```

Example: lme4



Javascript fragment

```
function fitModel(){
    Ext.Ajax.request({
        timeout: 60000,
        url: '../brew/lme-model',
        method: 'POST',
        params: {
            lmeRequest: Ext.util.JSON.encode(request)
        },
        success: function(o){
            try{
                result = Ext.decode(o.responseText);
            }
            updateParameters(result.modelFit);
        }
    });
}
```

lme Request

Request Object:

```
{
   "thisModelNumber":1,
  "reml":true,
   "dependent":"Reaction",
   "dataFile":95806,
   "DVfamily":"gaussian",
   "fixedIV":["Intercept","Days"],
   "randomIV":[{
     "group":"Subject",
     "varName":"113Subject",
     "predictors":["Intercept","Days"]
   }]
}
```

Server Fragment (R)

```
fitModel <- function(){
    lmeRequest <- fromJSON(as.character(POST$lmeRequest));

    reml <- as.logical(lmeRequest$reml);
    dependent <- lmeRequest$dependent;
    DVfamily <- lmeRequest$DVfamily;
    fixedIV <- lmeRequest$fixedIV;
    randomIV <- lmeRequest$randomIV;
    dataFile <- lmeRequest$dataFile;
    thisModelNumber <- lmeRequest$thisModelNumber;

    # do some parsing and then:
    load(datafile);
    myModel <- glmer(modelFormula, family=DVfamily, data=mydata);

    #extract information from myModel, and store it in a list
    return(toJSON(outputList));
}
```

lme Response

Response Object:


```
{
    success:true,
    modelFit:{
        "formula":"1 + Days + (1+Days|Subject)",
        "termlabels":["Days"],
        "dataClasses":{"Reaction":"numeric","Days":"numeric"},
        "fixef":[
            {  "id":"1",
            "name":"(Intercept)",
            "estimate":"251.405",
            "SE":"6.825",
            "tvalue":"36.836",
            "assign":"0",
            "factor":"Intercept"  },

            { "id":"2",
            "name":"Days",
            "estimate":" 10.467",
            "SE":"1.546",
            "tvalue":" 6.770",
            "assign":"1",
            "factor":"Days" }
         ], ... (continues)
```

lme Response continued


```
        "CorMatrix":[[1,0.066],[0.066,1]]}],
        "REML":"TRUE",
        "LL":-871.814,
        "Deviance":1743.628,
        "DF":6,
        "Sigma":25.592,
        "AIC":1774.786,
        "BIC":1755.628,
        "randomCovMatrices":[
            {  "group":"Subject",
            "fields":["(Intercept)","Days"],
            "CovMatrix":[[612,9.6],[9.6,35.1]]  }
        ]
    }
}
```

Some Advice

- Use Firefox + Webdev toolbar + firebug for development.
- Design a protocol that makes sense semantically.
- Don't use R to generate Layout or HTML. This soon become unmanageble.
- Only computing at the server, do all layout stuff at the client.

Connecting your HTTPD to R

RApache/brew:

- Plugin for Apache webserver + scripting package.
- New R session for every http request.
- Fast, good caching.

Calling Rscript from php/cgi

- No need for extra plugins.
- Works on windows.
- Extra sensitive to code injection!

Rserve:

- Keeps an R session alive.
- Requires additional Rserve client.
- If session gets messed up, you're screwed.

Security

- Watch out for code injection!
- Parse user data safely. Don't let a user specify which function to run.
- If necessary, remove any weird characters from argument strings.

Graphical Interfaces

- There are many free and open-source javascript libraries that provide ready-to-use interfaces and widgets.
- Easily create Grids, Tabs, Windows, Drag/drop, Trees, Forms, Toolbars, etc with only a few lines of code.

Framework	License
Prototype/Scriptaculous	MIT
Yahoo! User Interface Library (YUI)	BSD
jQuery	GPL or MIT
Dojo	BSD
Ext	GPL or Commercial

Thank you for your attention!

- Jeffrey Horner (2009). rapache: Web application development with R and Apache. http://biostat.mc.vanderbilt.edu/rapache/
- Hadley Wickham (2009). ggplot2: An implementation of the Grammar of Graphics. http://CRAN.R-project.org/package=ggplot2
- van Buuren S, Ooms JCL (2009). Stage line diagram: An age-conditional reference diagram for tracking development. Statistics in Medicine.
- w3schools: Full Web Building Tutorials - All Free.