Recent Developments in R

Paul Murrell

The University of Auckland

Tokyo 2003

S4 Object System

Object systems available in R

- R has two object systems available, known informally as the S3 and the S4 systems.
- S3 objects, classes and methods have been available in R since its inception. They correspond to the system described in Statistical Models in S (1990).
- S4 objects, classes and methods have been added recently. They correspond to the system described in Programming with Data (1998).
- Both systems are based on generic functions and method dispatch according to the class of one or more arguments.
- Many common functions in R are defined as (S3) generic functions.

Why use classes and methods?

Classes allow you to:

- Encapsulate the representation of an object (information hiding)
- Specialize the behavior of your functions to your objects
- Specialize the behavior of system functions to your objects

Information hiding

The major reason for using classes is to hide implementation details from the user. The user sees only the output from methods for print, summary, plot and other generic functions and doesn't need to know the internal structure.

This allows the developer to change the internal structure without requiring the user to make any changes.

Specialising behaviour

The grid package maintains a display list – a record of what has been drawn in an image. This contains many different sorts of objects (lines, circles, viewports, ...).

When it is time to redraw the image, the generic function grid.draw is called for each object on the display list.

Each sort of object has a grid.draw method which draws appropriate output.

Why S4?

S3 classes are very informal.

Consider an object of (S3) class lm:

```
> x <- 1:10
> y <- rnorm(10)
> lmobject <- lm(y ~ x)
> class(lmobject)
[1] "lm"
> names(lmobject)
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"
```

Methods written for this class rely on the object having the correct slots and on each slot having the correct information.

Why S4?

```
> summary(lmobject)
Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-1.2671 -0.7219 -0.2050  0.7407  1.6782

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.0786     0.7243   1.489    0.175
x            -0.1352     0.1167  -1.158    0.280

Residual standard error: 1.06 on 8 degrees of freedom
Multiple R-Squared: 0.1436,         Adjusted R-squared: 0.03654
F-statistic: 1.341 on 1 and 8 DF,  p-value: 0.2802
```

Why S4?

It is too easy to set the (S3) class of any object to be lm:

```
> notlmobject <- "This is not an lm object"
> class(notlmobject) <- "lm"
```

```
> summary(notlmobject)
Error in if (p == 0) { : argument is of length zero
```

Why S4?

An S4 class has its slots declared explicitly, the only way to create an S4 object is using the new() function, and only values of the correct type can be entered into a slot.

```
> setClass("demo", representation(x = "numeric"))
[1] "demo"
> new("demo", x = 1)
An object of class "demo"
Slot "x":
[1] 1
```

```
> new("demo", x="not a number")
Error in validObject(.Object) : Invalid "demo" object:
  Invalid object for slot "x" in class "demo":
    got class "character", should be or extend class "numeric"
```

Why S4?

S3 method dispatch is very informal; any function of the form a.b will act as a method for the generic function a and the class b

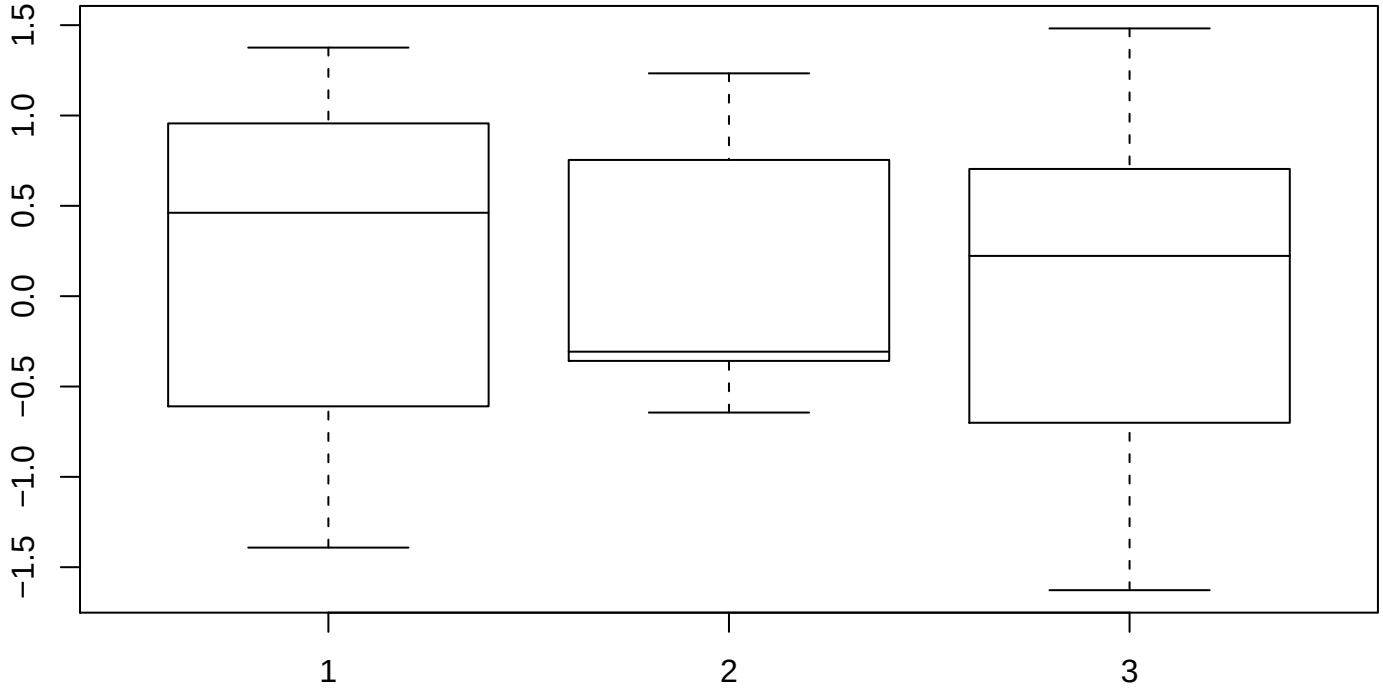
S3 methods only dispatch on the first argument.

```
> plot
function (x, y, ...)
{
    if (is.null(attr(x, "class")) && is.function(x)) {
      ...
    }
    else UseMethod("plot")
}
<environment: namespace:base>
```

Why S4?

plot.factor draws boxplots when the x argument is a factor.

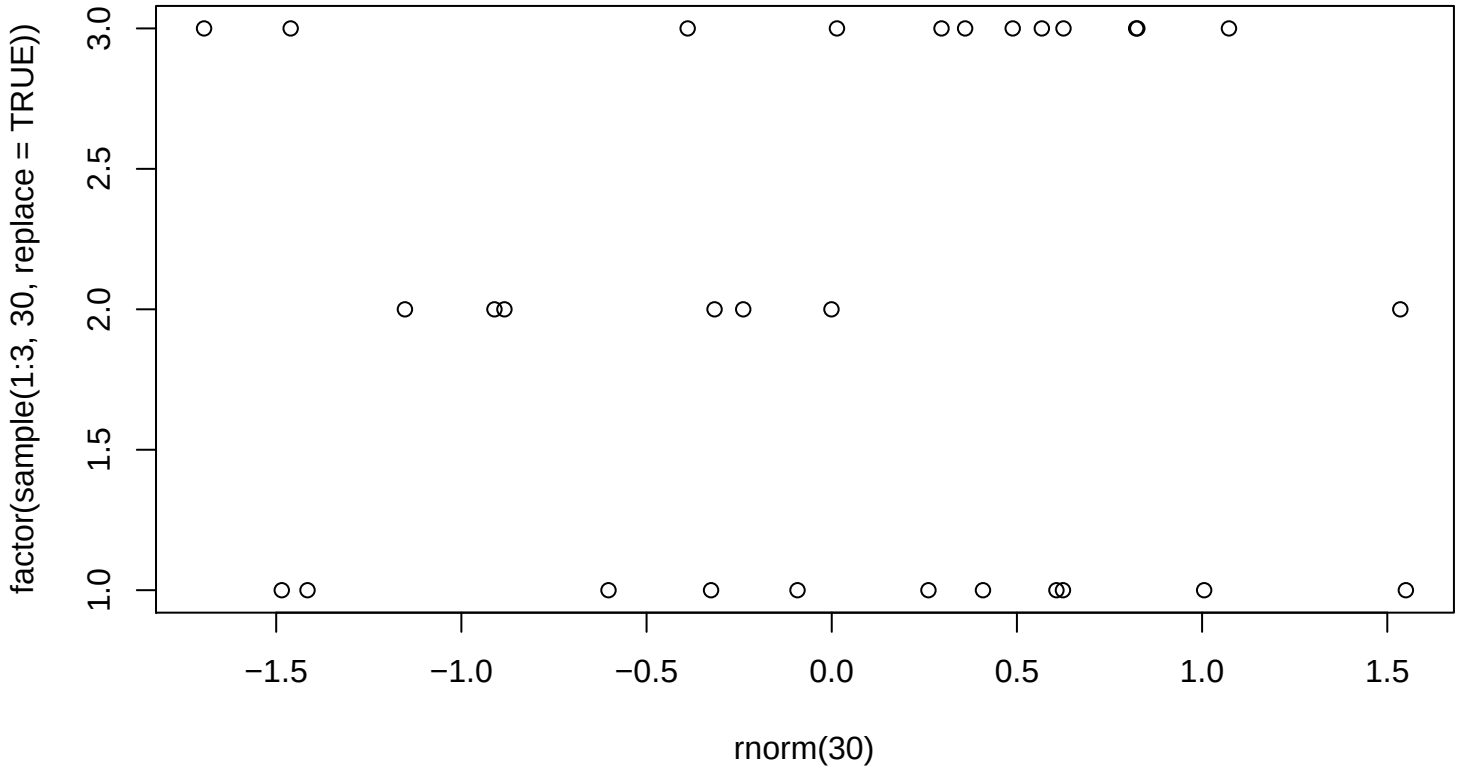
```
> plot(factor(sample(1:3, 30, replace = TRUE)), rnorm(30))
```



Why S4?

It would be nice if this produced a horizontal box plot.

```
> plot(rnorm(30), factor(sample(1:3, 30, replace = TRUE)))
```



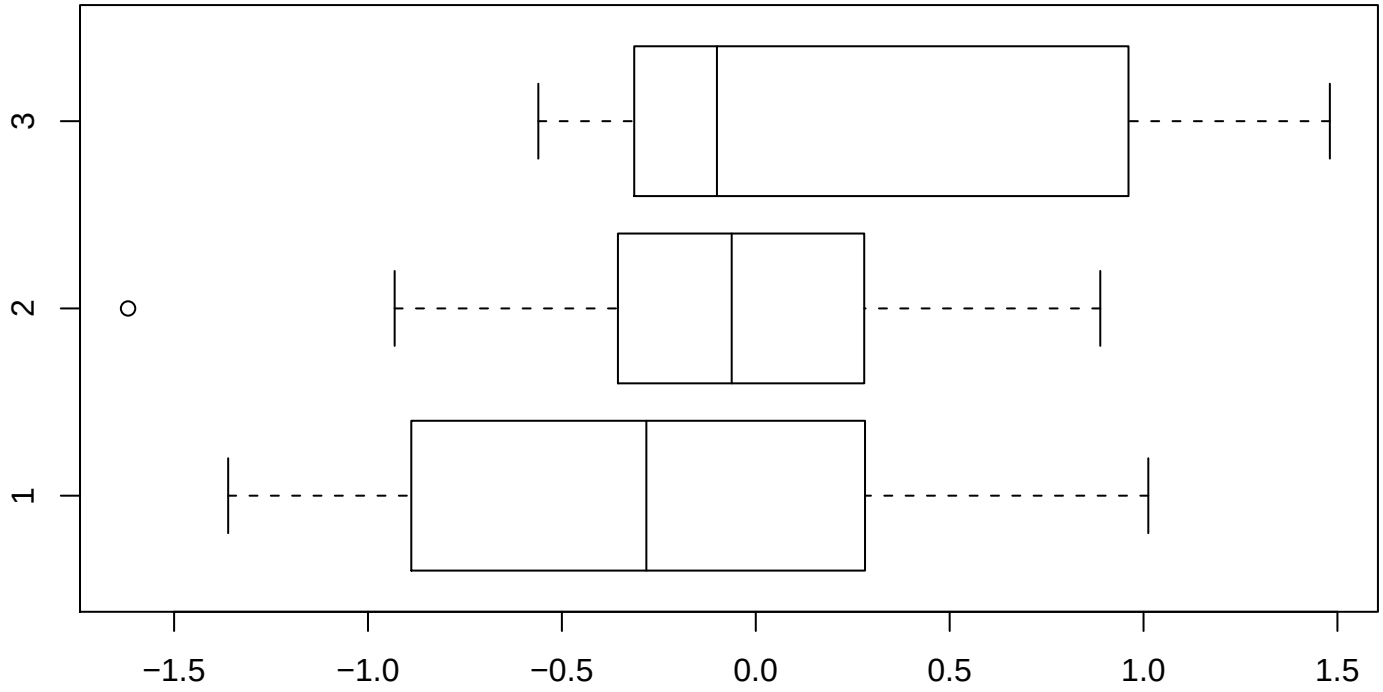
Why S4?

S4 methods are declared explicitly and dispatch occurs on all arguments.

```
> setMethod("plot", signature(x = "numeric", y = "factor"), function(x,
+     y, ...) {
+     boxplot(x ~ y, ..., horizontal = TRUE)
+ })
Creating a new generic function for "plot" in ".GlobalEnv"
[1] "plot"
```

Why S4?

```
> plot(rnorm(30), factor(sample(1:3, 30, replace = TRUE)))
```



Easing the transition from S3 to S4

When an S4 method is defined for an S3 generic function, the generic function is automatically coerced to an S4 generic function.

```
> rep
function (x, times, ...)
UseMethod("rep")
<environment: namespace:base>
> setMethod("rep", "demo", function(x, times, ...) {
+     rep(demo@x)
+ })
Creating a new generic function for "rep" in ".GlobalEnv"
[1] "rep"
```

Namespaces

Motivation for Namespaces

Consider the standard R search path:

```
> search()
 [1] ".GlobalEnv"      "package:tools"   "package:methods" "package:ctest"
 [5] "package:mva"     "package:modreg"  "package:nls"     "package:ts"
 [9] "Autoloads"       "package:base"
```

If we type a symbol at the command line, R searches this path, in order, until it finds the symbol.

```
> sum
function (..., na.rm = FALSE)
.Internal(sum(..., na.rm = na.rm))
<environment: namespace:base>
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
```

Motivation for Namespaces

It is very easy to shadow a symbol by defining a symbol higher up the search path.

```
> sum <- function(x, y) {
+     x + y
+ }
> sum
function (x, y)
{
    x + y
}
```

Namespaces make it possible to define a symbol, but not put it on the main search path.

Motivation for Namespaces

An R package usually contains many functions, but only some of them are intended for the user to call.

```
addTwo <- function(x, y) { x + y }

sumTwo <- function(x, y) {
  x <- as.numeric(x)
  y <- as.numeric(y)
  addTwo(x, y)
}
```

Namespaces make it possible to hide a symbol from the user.

Description of a Namespace

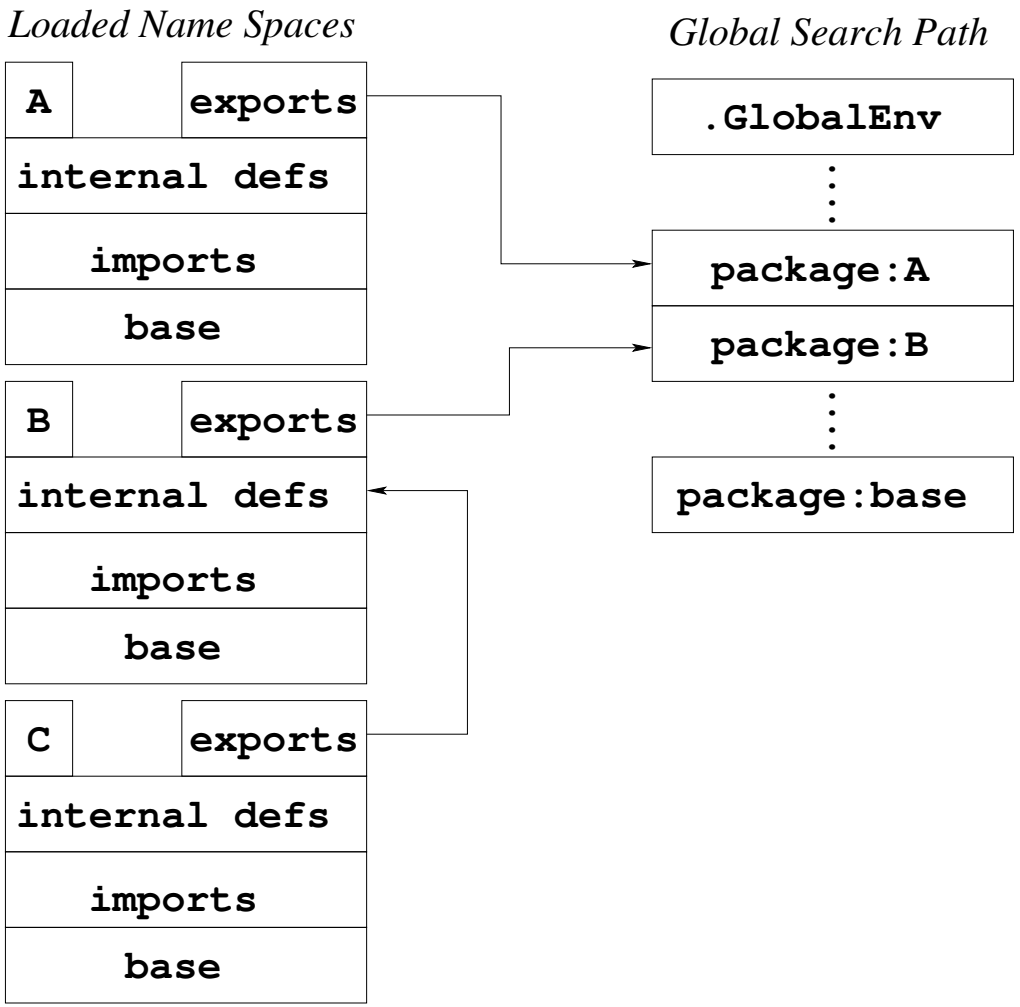
A namespace affects an entire package. The namespace consists of a declaration of the symbols that the package exports.

```
export(sumTwo)
```

A namespace can also contain a declaration of the symbols that the package imports from other packages.

```
import(mva)
```

Namespaces and the Search Path



Namespaces and Objects and Methods

S3 methods must be registered so that they can be seen from outside a namespace.

```
S3method(print, trellis)
```

S4 classes and methods are explicitly exported using special declarations.

```
export("mle")
exportClasses("mle", "profile.mle", "summary.mle")
exportMethods("confint", "plot", "profile", "summary", "show")
```

Shortcuts and Cheats

The :: syntax can be used as a short-hand for importing a variable from a namespace. This can be useful for specifying a symbol that may be shadowed.

```
> base::sum
function (..., na.rm = FALSE)
.Internal(sum(..., na.rm = na.rm))
<environment: namespace:base>
```

The ::: syntax can be used to find a symbol that is hidden in a namespace. This can be useful for debugging code or viewing the source code for a function in a namespace.

```
> library(lattice)
> lattice:::print.trellis


function (x, position, split, more = FALSE, newpage = TRUE, ...
  ...
<environment: namespace:lattice>
```

References

The section on "Package name spaces" within the section on "Creating R packages" in the "Writing R Extensions" manual.

"Name Space Management for R" by Luke Tierney, in *Rnews* Volume 3(1), 2003.

Documentation Vignettes

Definition of a Vignette

A piece of documentation on a particular topic that gives a more detailed description than a help page, but less than a book.

Currently, R supports vignettes which are produced from Sweave documents in PDF format.

What is Sweave?

Sweave is a system for writing live documents that contain both text (LaTeX) and S code. When the document is processed by the Sweave() function in R, the code is extracted and run and the output is inserted back into the document.

- Readers know that the output is accurate and reproducible
- Input, output and text can be freely mixed.
- The document can be processed again to update the data, or as a regression test for the underlying software.

Sweave is in the tools package in the R distribution. It has uses apart from documentation: these slides are an Sweave document.

Sweave Document Structure

An Sweave document is divided into text chunks and code chunks. Code chunks begin with <<options>>= and end with @ The text chunks should form a valid LaTeX file, eg:

```
\documentclass{article}
%\VignetteIndexEntry{SweaveDemo}
\title{An Sweave Document}
\usepackage{Sweave}
\begin{document}
\maketitle
Here is a plot:

<<fig=TRUE>>=
plot(1:10)

@
\end{document}
```

Sweave commands

A code chunk is sent to R. The options in the chunk header control what goes back in the document in a LaTeX Schunk environment

- echo=TRUE/FALSE to display the input code.
- results=show/hide to display the output from the code.
- fig=TRUE to include the graphical output from the code.

Up-to-date manual at `http://www.ci.tuwien.ac.at/~leisch/Sweave`

What goes in a Vignette

It should be short and explicit.

- It should be about a single topic.
- It should contain runnable code and rely on data that are available in R or the libraries needed to carry out the task being documented.
- Vignettes should not be about single functions. The function documentation is the right place to document that. Vignettes should document a process or task and will typically involve several functions.

Creating a Vignette

```
> library(tools)
> Sweave("SweaveDemo.Rnw")
Writing to file SweaveDemo.tex
Processing code chunks ...
 1 : echo term verbatim eps pdf

You can now run LaTeX on SweaveDemo.tex
```

```
bash$ pdflatex SweaveDemo
```

... OR ...

Put the .Rnw file in `pkg/inst/doc`

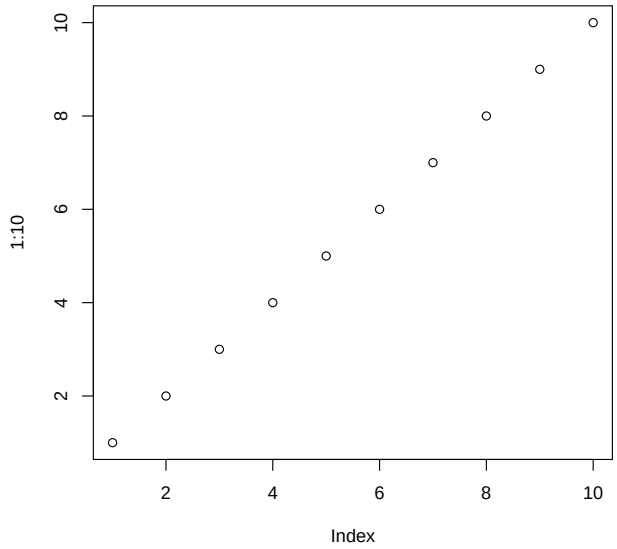
... THEN ...

```
vignette("SweaveDemo")
```

The SweaveDemo Vignette

Here is a plot:

```
> plot(1:10)
```



Compiling R

Opportunities for Speed Improvements

- Looking up symbols (Namespaces, sealing environments)
- Constant values (constant folding)
- Evaluating expressions (byte-code compilation)

Examples

Luke Tierney is working on a compiler for R which can already halve the execution time of some code.

```
> library(compiler)
> f <- function(x, mu = 0, sigma = 1) {
+     (1/sqrt(2 * pi)) * exp(-0.5 * ((x - mu)/sigma)^2)/sigma
+ }
> fc <- cmpfun(f)
> x <- seq(0, 3, len = 5)
> system.time(for (i in 1:1e+05) f(x))
[1] 1.91 0.00 1.91 0.00 0.00
> system.time(for (i in 1:1e+05) fc(x))
[1] 0.82 0.00 0.82 0.00 0.00
```

Examples

It is also possible to compile entire files and packages. However, if a package contains a large amount of C code, the results are less dramatic.

- Compiling the grid package only produces approximately 10% speed up.

References

Luke Tierney's "Notes on Compilation in R"
`http://www.stat.uiowa.edu/ luke/R/bytecode.html`

Other Things

Database Integration

There now exists a standardised interface for communicating between R and various RDBMS. This is implemented in the DBI package, with implementations of the interface for MySQL (RMySQL), Oracle (ROracle), and SQLite (RSQLite).

http://stat.bell-labs.com/RS-DBI/index.html

There are also packages providing specialised access to specific RDBMS – PostgreSQL (RPgSQL) and mSQL (RmSQL) – and another standard interface for ODBC (RODBC).

Event loops and GUIs

There is still no single, official GUI for R.

There are two main cross-platform toolkits supported: tcltk (Peter Dalgaard) and Gtk (Duncan Temple-Lang). R-(D)COM (Thomas Baier and Eric Neuwirth) provides a path for Windows development.

John Fox has produced a reasonably general (tcltk-based) GUI encompassing introductory-level statistics (Rcmdr).

Phillipe Grosjean maintains a site with links to the various GUI efforts (including his own SciViews).

Work still needs to be done to allow R to become properly event-driven.

Threading and Parallel Computing

R is NOT thread-safe.

It has been recently stated that user-level threads in R would be "trivial".

Many obstacles still exist to making R's internal computations use threading (lack of a cross-platform native thread implementation, making R's C code safe, making contributed code safe, ...).

Several solutions exist for "embarassingly parallel" computations (see Luke Tierney's snow package).

grid Graphics

R now has two distinct graphics systems: the "traditional" S graphics system (what I call base graphics); and the new grid system.

To learn more about why the grid system exists and how it works, please come back tomorrow!