Connecting Scientific Software

Duncan Temple Lang



www.omegahat.org
duncan@research.bell-labs.com

August 6, 2003

The State of Statistical Software
Social

- Our work is experiencing amazing growth in “importance”.
- Statisticians and non-statisticians.
  - Computationally intensive methods
  - Inter-disciplinary information technology.
- Opportunity to realize potential of delivering statistical content.
- Real need for infrastructure to bring us through the next decade.
- Relatively little research in statistical computing
  more methodologies that use computing
  very few in *fundamentals*.

- Challenge to make statistical computing a respected component of statistics research.

- Academic community must support and prioritize teaching, and fostering and rewarding work in this area

- And *we* must avoid label of “just programming”
  need high-level of scholarship.

The State of Statistical Software
Technical

- R is becoming very popular across fields.
- R (and Omegahat) provides rich medium for disseminating statistical methodology.
- Based on S model
  evolved over last 20 years.
- Last 5 years, Information Technology world exploded with new technology and paradigms.
- Will S suffice?
  Do we need a new system(s)?
  Build new layers on top of existing functionality?

Short- to Middle- term

- Share software across different disciplines.
  - Incorporate new software from other fields into S.
  - Export statistical methodology to non-statisticians.
- Vital Constraint:
  Must be done so that each community works in their familiar environment and need not learn new tools.
- Access R via
  - functions/macros within a spreadsheet
  - from Matlab, Perl, Python, Java, Objective-C, Visual Basic.
  - via SQL calls (i.e. embedded functions)
- Humans don't learn two languages
  To computers, its all bits!

- Can connect systems without users of one knowing the other language.

Inter-System Interfaces: Essential Features

- Work at level of function calls, not language/syntax.
- Bi-directional
  call R from X, and X from R
- Convert primitive objects to primitive objects
  i.e. **character** in S to string in Python.
  arrays of same
- Complex data types transferred as references
  stays in native system
  can be used in subsequent calls

Example: R from Python

```
data = R.dataFrame(x = [1.2, ...], y = [2.4, ....])
fit = R.lm("y ~ x", data)
R.coefficients(fit)
```

- data and fitted model stay in R where they makes sense.
- hold (anonymous) reference to them in local language.

Existing Interfaces

- Works for
  - R-Java (SJava)
  - R-Perl (RSPerl)
  - R-Python (RSPython)
  - R-*Objective-C* (RSPerl)
  - R-JavaScript (SNetscape)
  - R-Octave (ROctave)
  - R-XSL (SXSLT)
  - R-Postgres (REmbeddedPostgres)
  - R-XLispStat (RXLisp)
  - R-S-Plus (RinS)

R-DCOM

- DCOM interface
  Distributed Component Object Model

- Two worlds:
  - R as a DCOM Client (RDCOMClient)
  - R as a DCOM Server (RDCOMServer)

- Distributed computing across
  - components/applications
  - machines

- DCOM is everywhere in Windows
  Excel, Word, PowerPoint, ADO, Browser, . . .

- CORBA is the equivalent in Unix world.

R as a DCOM Client

- Control other applications from R with “regular” R functions.
- Create COM object

```
excel1 = COMCreate("Excel.Application")

ie = COMCreate("InternetExplorer.application")
```

- Call methods

```
excel1$Open("C:\\data.xls")
ie$Navigate2("http://www.omegahat.org/")
```

- Access properties/fields

```
book = excel1[["Workbooks"]][[1]]
sheet = book[["ActiveSheet"]]
```

- Using higher-level interface in R

```
sheet[1,1] <- matrix(rnorm(100), 10, 10)
```

Client Mechanism

- User calls methods as local S functions.
- R does not know about the COM method ahead of time.
- R passes data from R to COM method as arguments.
- Small kernel of C code to
  - lookup method
  - convert values to DCOM types
  - perform call
  - convert result back to R.
- Works generically for any DCOM object.
- Convert complex R object as COM object....

Discovering the Methods

- How do we find out what methods are available?
- Read documentation.
- Visual object browser tools, e.g. Visual Basic.
- Ask the object from R
  reflection.
- SWinTypeLibs package to read information about classes, methods, parameters and return types.
- Can use this to compile access to methods.
  Run-time versus compiled lookup.
- Provide information for how R objects are converted to COM server objects.

Example

```
library(SWinTypeLibs)
f = "C:\\Microsoft Office\\Office\\EXCEL9.OLB"
lib = LoadTypeLib(f)

names(getFuncs(lib))
```

R as a COM server

- Can make command line/interpreter accessible.

```
cmd = "myData <- data.frame(x = c(1.2, 3.19, ..),
                            y = c(8.5,  12.21, ...)"
R.evalString(cmd)
R.evalString("fit <- lm(y ~ x, data)")
```

- Done by Luke Tierney for XLispStat, Thomas Baier for R.

- Challenges
  - User must know 2 languages - theirs and R.
  - complicated to put local data into commands
    e.g. values for x and y.
  - manage global variables
  - debugging is complex

- can’t express all computations.

S-Plus Approach

- Export individual functions

```
f = CreateObject("S.rnorm")
f.setParameterTypes("vector,vector")
f.setParameter("n", 10)
f.setParameter("mean", 100)
f.evaluate()
values = f.returnValue()
```

- Awkward interface for programmer
  not simple function call

```
f = CreateObject("S.rnorm")
vals = f(10, 100)
```

Alternative Approach

- Export COM objects from S
  with methods and properties
- To client, looks like regular COM object.
- From R, just collection of S functions.

```
library(RDCOMServer)

funcs = list(ttest = function(x, ...)
                       t.test(as.numeric(x),...),
             rnorm = rnorm)

def = SCOMFunctionClass(funcs, name = "R.TTest")

registerCOMClass(def)
```

- Client uses this as

```
t = CreateObject("R.TTest")
t.ttest([1.2, 3.2, -2.1])
t.ttest(t.rnorm(100, 10))
```

- Share state and persistent properties using closures/lexical scoping.

- R command line is a simple example.

R interpreter server

```
els = list(
  get=get,
  set = function(name, value, pos = globalenv(), ...) {
            assign(name, value, pos = pos);
            TRUE
          },
  exists = exists,
  evaluate = function(cmd) {
       e <- parse(text = cmd)
       eval(e)
  },
  library = library,
  search = search,
  print = function(x, ...) {print(get(x), ...); NULL},
  call = function(name, ...) {
                do.call(name, list(...))
```

```
        },
    objects = function(name = 1, ...) {
                    objects(name = name, ...)
                })

def <- SCOMFunctionClass(els, name = "R.Evaluator",
                         def=new("SCOMEngineClass"))

registerCOMClass(def, profile=fileName)
```

How does it work?

- Store functions and UUID as R object.
- Store UUID and name in Windows registry.
- When client creates object, COM asks R with UUID.
- R gets corresponding functions
- Creates C++ object with those functions
- Client calls method, C++ object calls corresponding function.

Event Servers

- Listen for events from applications
  creating new workbook or sheet in Excel
  editing a cell, selecting a row
- Register COM server with event source
- Method called when event occurs.

```
f = "C:\\Microsoft Office\\Office\\EXCEL9.OLB"
lib = LoadTypeLib(f)
s = createCOMEventServerInfo("IAppEvents", .lib = lib)

s$methods$SheetSelectionChange =
function(sh, range) {
  which = seq(range$Row(), length = range$Rows()$Count())
  plot(data[which,])
}

server = createCOMEventServer(s$methods, s$ids)
connPoint = getConnectionPoints(book)[[1]]
cookie = connectConnectionPoint(connPoint, server)
```

- Leveraging DCOM infrastructure.
- RDCOMClient - accessing COM from R.
- RDCOMServer - export R functions to other applications.
- SWinTypeLibs - reading meta-information from COM objects
- SWinRegistry - accessing Windows registry.
- Ruuid - Robert Gentleman's Universal Identifier package.
- excelUtils - David James' high-level package for accessing Excel from R.
- Much credit to David James.
- http://www.omegahat.org/