The State of Statistical Software

Duncan Temple Lang
Statistics and Data Mining Group
Bell Labs, Lucent Technologies

August 14, 2003

If we review where we are now, the state of Statistical computing seems good. There are two primary groups of users its serves: statisticians and non-statisticians. Looking first at non-statisticians, good things are happening now. As data-related technology (e.g. Web Services, IT) and applications (e.g. BioInformatics) are experiencing massive growth, the need for data analysis is experiencing a related surge. Importantly, data analysis is becoming an essential part of the decision making *process* rather than a separate "off-line" activity carried out by statisticians. Instead, we are challenged to deliver our cutting-edge methods to non-statisticians as part of *their* standard software.

For statisticians, the growth of an Open Source community and the support and advancement of R promises good things for facing the challenges of statistical research and performing complex, flexible data analysis. Importantly, it has included many in the development of software and encouraged statisticians to contribute by providing and disseminating statistical ideas and methodology via software. This is something that has been lacking in other disciplines and which need not necessarily be done as effectively if we rely on commercial software.

So, so far so good; however there are some serious issues to be dealt with. As we move to increasingly large datasets and more computationally intensive methods, the model of our present highly flexible systems will likely fail to meet the needs of statisticians. And to realize the potential of delivering statistical methodology to non-statisticians, we must be able to make our software easily and readily available directly within their software (Excel, Databases, etc.) and programming languages (Perl, Visual Basic, etc.). The S environment was originally conceived and designed about 20 years ago. SAS computational model, I believe, is of the same vintage. SPSS, Minitab, etc. are a different species, and while Matlab is becoming a highly used tool in statistical research, it does not lend itself as well to expression of statistical ideas or re-usable, publically distributable software development.

We should congratulate "ourselves" (whoever that exactly may be?) on how much we have accomplished and how good our software systems are for support data analysis, and experimenting with new ideas and methodology. At the same time, without wanting detract from the excellent work that has and is being done and without being alarmist, I think we should note that our user base is rapidly growing more diverse and the information technology with which we must interact with these users is growing even more rapidly. We need a great deal more customizable, extensible and generally flexible software that can be used in a wide variety of circumstances. We need to find the correct infrastructure and primitives and allow people to develop different languages, different data models, etc. on top of this to suit their evolving needs. We have some serious challenges ahead and we need some serious work to be done. And the academic community should not expect this to be done by commercial/industrial entities. They must actively encourage research within our own community, and foster young talent.

And that brings us to a reasonably serious problem in the "state of statistical computing". Where are the people that will work on this increasingly important topic? Looking at web pages of many (academic) statisticians, many list statistical computing among their interests. However, there are relatively few institutions teaching modern statistical computing and information technology to statisticians. Few are doing research in statistical computing of systems, but more in algorithms and methodology that *use* a computer. JCGS is made up primarily of classical statistical methodology that is done on a computer. And this is all good and useful to our field and others. However, there are very few people who are working on what one might term "paradigms for statistical computing". And this is a very important area as the I.T. world explodes around us. The development of systems, languages, libraries and general infrastructure for computing with data is done by a handful of people around the world. And because of the complex, specialized nature of systems like S (R and S-Plus), we can expect this to basically persist. There are relatively few people who can invest the time in learning the details of the R engine. We need a more extensible, adaptable "system"

not only for new and varied applications but to facilitate and encourage people to experiment with it and be able to try new paradigms. For example, it is prohibitive for an enthusiastic researcher to try to add a predictive algorithm to control R itself (e.g. cache management of variables). Figure 1 displays a simplified view of different levels of computer sophistication or specialization. What we see is that almost everyone in our discipline benefits from good computational research tools for performing simulations, developing methodology or performing data analysis. And almost all benefit from high-level systems such as S, Matlab and SAS. But the number of people working in these latter areas is very small. Far too much is left to commercial entities that must respect a different market.

So how do we secure the future of statistical computing and get good people to work in this area? We face a real problem that statistical computing is not adequately embraced and rewarded by more traditional members of our community. It is seen as simply programming and few can make a career focusing intensively on this area. This is to a large extent based on an lack of understanding of what statistical computing is and how design, architecture and paradigm development differs from simply programming. However, "we" in the statistical computing community also are to blame. There are a great deal of people who advertise parts of their research as statistical computing, yet it amounts simply to writing programs to do a specific task. There is little effort made to generalize, connect to other systems, reference, compare and use existing software. Generally the level of scholarship in statistical computing is quite low; many projects are merely competent Masters-level theses. There is a confusion between useful and popular software with that of good structure and thought. As we educate people to appreciate our discipline, we must also insist on decent scholarship and, while encouraging and rewarding effort in programming, differentiate between research and programming.

Having espoused this somewhat provocative perspective on the current state of statistical software, let me discuss an area that I think is of non-trivial importance in both the short– and long–term. We at Bell Labs have been focusing for several years on the connectivity problem I mentioned above. We have developed a general model that allows statisticians to export software from S to "users" or consumers who need not be in any way familiar with S. And the same model allows statisticians to easily access functionality from other languages and systems without having to understand those languages in which that functionality is implemented. Essentially, we are getting to the point where people can program in their favorite language and share it with others in their favorite language. This connectivity is important for a variety of reasons:

- to transfer statistical methodology to scientists,
- to take advantage of methodology and software from other disciplines,
- to be able to do both of these readily in the future without tedious, manual labor,
- to be able to introduce new languages and environments for doing statistical computations while leveraging our existing tools

One of the original motivations for S was to provide a high level language with which researchers and data analysts could call existing Fortran routines. The language grew essentially into a shell with which users could input data, pass it to a sequence of Fortran routines, save the answers to variables at intermediate points and do interactive exploratory data. This was a very big deal at the time and one that has become ingrained in our work practices now. After Fortran, the interface to C and the operating system commands via the shell were added. These effectively allowed users to program in one language and access functionality in the other. This was a form of interactive, high-level C/Fortran programming. Data from S was passed to C, and results came back and stored as variables for use in subsequent computations. One didn't need to know the C or Fortran languages, just the number and type of the arguments of the routine(s) of interest. (Of course, it was a lot more thorny than this in practice and few were without the scars to prove their worth!)

In this era, there are many other languages in which people program all sorts of useful material. Perl, Python, Java, Matlab/Octave, Objective-C are just some. Each is specialized to different domains and tasks. It is hard to beat Perl's expressiveness and speed for doing text manipulation on lines of data. Java is very useful and popular language for a variety of tasks, including lower-level numerical computations. Matlab and Octave are extremely useful high level languages for matrix operations and engineering-oriented computing. Users of one system rarely invest time in learning the other. There is a large overhead and shifting between them and learning and remembering mundane things such as debugging tricks, etc. are prohibitive. But the result is that we either don't use ideas developed in these other languages, or we recode them, and often erroneously. Similarly, engineers, biologists, etc. using Perl, Java, and so on

can't access good or modern statistical software and methodology because it is written in, what is to them, an arcane language.

The challenge is simple. We must find a way to allow statisticians to access code in other languages and make it seem like a natural part of their familiar language. In other words, we must be able to call Java, Perl, Python, Matlab functions/methods as if they were regular S functions. And if we are to realize the potential of statistics in different scientific disciplines, we must make our statistical software available to non-statisticians in their familiar environments, be it languages like Perl, Python, Matlab or applications such as spreadsheets (e.g. Excel or Gnumeric), databases, Web services or Web browsers. Well the good news is that we can do this, and we can make the interface on both sides almost seamless.

We have implemented these bi-directional bridges or inter-system interfaces between R and Perl, Python, Java, Objective C, Octave, Netscape (and JavaScript), Postgres (R inside Postgres), Gnumeric, CORBA, and now recently DCOM on Microsoft Windows. Since the DCOM interface is new and since it is very generic, we will talk about this in the remainder of the talk and show how we can a) access other COM applications from R such as talking to Excel, Internet Explorer, Acroread, Word, and a multitude of software components running on Windows; and b) make collections of statistical methods available to other applications which can be used in Visual Basic, Perl, Python, C++, etc.

The key behind this is that S users don't need to know much about COM. They can create COM objects from S (using the function *COMCreate()*).

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

The resulting value of *ex* is a reference to a COM object that (usually) lives in a separate process. However, it appears to be local to R and we can call any of its methods. For example, we can ask Excel to open a file

```
ex$Open("C:\\myData.xls")
```

and then we can ask it for the active worksheet within the workbook

```
sheet =  ex[["Workbooks"]][["ActiveSheet"]]
```

Here we are accessing fields or *properties* of the COM objects, first the Workbooks and then the ActiveSheet within the Workbooks collection.

Writing the data to a file and asking Excel to read it is a little clumsy. Firstly, we must synchronize the two applications (R and Excel) and then clean up the file at the right time. This can be non-trivial. But more importantly, we lose information. If we have an integer in R, say 1, it will be read as a real number in Excel. While this is probably trivial, more complex objects cannot be transferred to Excel without losing information. So, of course, there is a better way to send data from R to Excel. We can put the values directly into a sheet from R. We ask the sheet for a reference to a specific Range which is handed to R as another COM object. This provides methods for setting its cells with values. We can make this look more S-like by providing a method for the **[¡-** method in S for an Excel Worksheet. Then the R user can use commands of the form

```
sheet[1,1] <- data.frame(x = 1:100, y = 10 + 2*(1:100) + rnorm(100))
```

David James has put together a package (excelUtils) that does this. Eric Neuwirth has one based on Thomas Baier's R(D)COM package.

Note that the communication is much more like calling an S function or a C routine from S. There is no indirect mechanism to first transfer the arguments. The result comes back as a regular S object. And no information is lost. The arguments are converted from S to their DCOM representations when the call is made. This works in a rather obvious manner. Strings in R are mapped to strings in DCOM, similarly, numeric, integer and logical vectors are mapped either to the equivalent scalars or to arrays of those types. Similarly, ordinary lists with no names are converted to DCOM arrays. S objects with a class (or lists with names) are transferred as COM objects themselves. Rather than losing information, we turn such objects into regular COM servers that other applications can access and query fields and invoke methods. This bi-directional and symmetric design is why this model works so generally. R is both a COM client and a COM server. And it exports server objects with methods and fields rather than via the R language. In this way, we can use complex objects in R as arguments to COM methods. We control how they are turned into COM objects, and can define their methods and properties entirely within R. There is no C code needed.

We'll return to servers a little later. In the meantime, let's think a little about how this connectivity works so that we, in S, can invoke methods in COM objects. The answer is quite short and simple and provided for us by COM's infrastructure. COM provides a form of dynamic invocation which allows us to create a call to a method at run-time. There is no need to compile and dynamically load C code into S for each COM type/class. Instead, we need only a simple generic mechanism provided by the RDCOMClient package to access any COM object method or property, now or in the future.

There are two natural questions that might cross ones mind. Firstly, how can we find out what methods are available? And secondly, isn't this dynamic invocation mechanism slow like it is for interpreted languages like S, Python, Matlab? Related to both of these questions is a concept called reflectance. Most COM classes come with a full description of their methods and properties. From our perspective, the important aspect is that this is programmatically accessible. The SWinTypeLibs package allows us to read this "type" information for a COM class (and other things such as DLLs). In this way, the S user can interactively browse the available methods, the number and types of the arguments, the type of the return value, etc. There are also interactive GUIs available in Excel, Visual Basic, etc. that present the information conveniently. And of course, this is a similar problem of discovering what routines are interesting in a Fortran or C library. We can always use the low-tech solution and read the documentation!

We can also use this type information from within S to programmatically generate S and C/C++ code that allows us to access the COM methods and properties. All of the necessary information is available to us and this would remove the need to query it at run-time. This should make things quicker, although I have not gathered any data yet. My expectation is that we are spending most of our time in the computation itself and in the latency of network communications while connecting to the remote COM objects. As a result, we would only be optimizing a minor part of the overall time. But the important thing is that we can use this reflectance information in a variety of ways, including catching errors (e.g. incompatible argument types) on the client side before calls are dispatched to the (remote) server and providing the user with more information. The SWinTypeLibs package is a powerful addition to our COM infrastructure.

We have mentioned that we can create COM server objects in R. Let's revisit this in more detail. The mechanism is quite simple. A COM class consists of methods and properties. We can create individual instances of such a COM class and each will have its own individual versions of these properties and methods. When we map this to S, what we want is a collection of variables that represent properties and a named list of functions. Suppose we wanted to provide an R COM server to fit a (linear) model so that other applications could get good statistical facilities without developers having to write them themselves in Visual Basic, for example. What do we need? Three methods should be adequate for a simple start: fit(), residuals() and coefficients().

In R, we might define this as

```
lmCOMGenerator =
function() {
 model <- NULL
 coefficients <- NULL

 fit = function(y, x) {
   model <<- lm ( y ~ x, data = data.frame(x = x, y = y))
 }

 list(fit = fit, coefficients = function() coef(model),
      residuals = function() residuals(model))
}
```

When the client calls *fit()*, we store the fitted model in the variable *model*. Then we can access it in the other methods to compute the coefficients and residuals using standard S functions. The idea here is that this is a generator or constructor function. When R is asked to create a COM object of this type, it calls this function and uses the return value as the basis for the COM object. The return value in this case is a list of functions that are the methods for the COM object. When a client calls one of the methods, the low-level COM server object in turn calls the associated S function.

That is the hard part of our COM server in R. We need only make it accessible to clients so that they can create an instance of this class. We do this by storing the definition of the class in R and adding an entry to the Windows registry where COM looks for COM class definitions. The S commands are as follows:

```
lmDef = SCOMIDispatch(lmCOMGenerator, name = "RLinearModel", help = "Fits a linear model")
registerCOMClass(lmDef)
```

We first tell R how to create the appropriate COM object using the generator function and we give it a name by which it is known in both R and by clients. Then we are ready to insert it into R's collection of COM class definitions and the Windows registry. The latter is done using the SWinRegistry package that allows us to read and write the Windows registry from R. This is just one use of it.

At this point, the class is defined and clients can access it as needed. For example, in Python, we might have some code that uses it something like

```
x = [10, 12, 13, 14]
y = [9, 11, 12.5, 16]
lm = win32com.Dispatch("RLMServer")
lm.fit(x, y)
lm.residuals()
lm.coefficients()
```

The `RDCOMServer` package provides a variety of different ways of defining servers in R. One can use generator functions, a simple list of functions, S4 methods, etc. The flexibility is not important at this point except for to note that it allows S programmers to think primarily in S.

So why are we talking about this topic as if it were new work? Aren't there already ways to use R and S-Plus as a COM server? Indeed there are, but the model they present have certain weaknesses when one considers how clients access them. Remember that we want non-statisticians to be able to access S functionality *as if it were in their own familiar environment/language*. Thomas Baier's R(D)COM server makes the R interpreter and language accessible to COM clients. These clients send S commands as strings to the server. This means that clients must know the R language, construct potentially complex S command strings by inserting data from their application into the sting, and also manage variables within S. If the command produces a result, it must be assigned to a variable within the command and then that S variable accessed in subsequent commands. For example, to use the linear model example above, a Python client would have

```
n
R.evalString("myFit <- lm(y ~ x, data.frame(x = c(10, 12, 13, 14), y=c(9, 11, 12.5, 16)))"
R.eval("residuals(myFit)")
```

Creating the first string involves converting the arrays into appropriate strings and inserting them into the overall command. And how would we pass a reference to a Python object (e.g. a COM object in Python) as an argument? We simply cannot do this. So the set of feasible computations is limited. Luke's XLisp version employs a similar model.

Note also that we have to provide a name of a variable in S with which to store the result. What if this is called twice by Python; one will overwrite the other. This is a potentially serious problem in an evaluation model such as a spreadsheet where formulae in different cells are evaluated in a non-obvious order.

S-Plus provides COM server facilities also and does it in a different way. One can make individual functions available as COM servers. A client calls them in the following rather awkward manner:

Our model makes the servers appear as regular COM objects. It hides the S language by default, but of course one can define a server entirely within S that evaluates commands. So the S-language server is just a special case of our model. Importantly, our model supports bi-directional communication and is extensible. Non-primitive objects in S are returned as generic COM objects with fields. S programmers can define S methods for creating customized COM objects from different S data types.

A final addition to this infrastructure follows naturally from being able to define general COM servers in S. We can create S objects that listen for events from COM servers. For example, we might ask Excel to notify us when somebody selects a row or column. A particular method of the S server is then invoked by Excel indicating this event and it can do whatever it wants. We can use this in a variety of ways. For example, we can implement a simple linked plot example where the user selects a row and we highlight the associated point in a plot. Or we can perform subsetting this way. The event mechanism is again a general COM concept and we can listen for events from arbitrary COM event sources such as Web browsers, Data objects (Active Data Objects - ADO) and so on. These event handlers in S are simply specific types of COM servers, created in the same way as a regular server.

This combination of client, server and event handlers gives us all the flexibility we need to implement quite sophisticated and powerful applications. We are using this infrastructure to make sophisticated statistical methodology available to some of our engineers in Lucent Technologies in studying production yield of optical packs. The SWinRegistry and SWinTypeLibs packages provide the necessary support to automate much of this software development entirely within S.