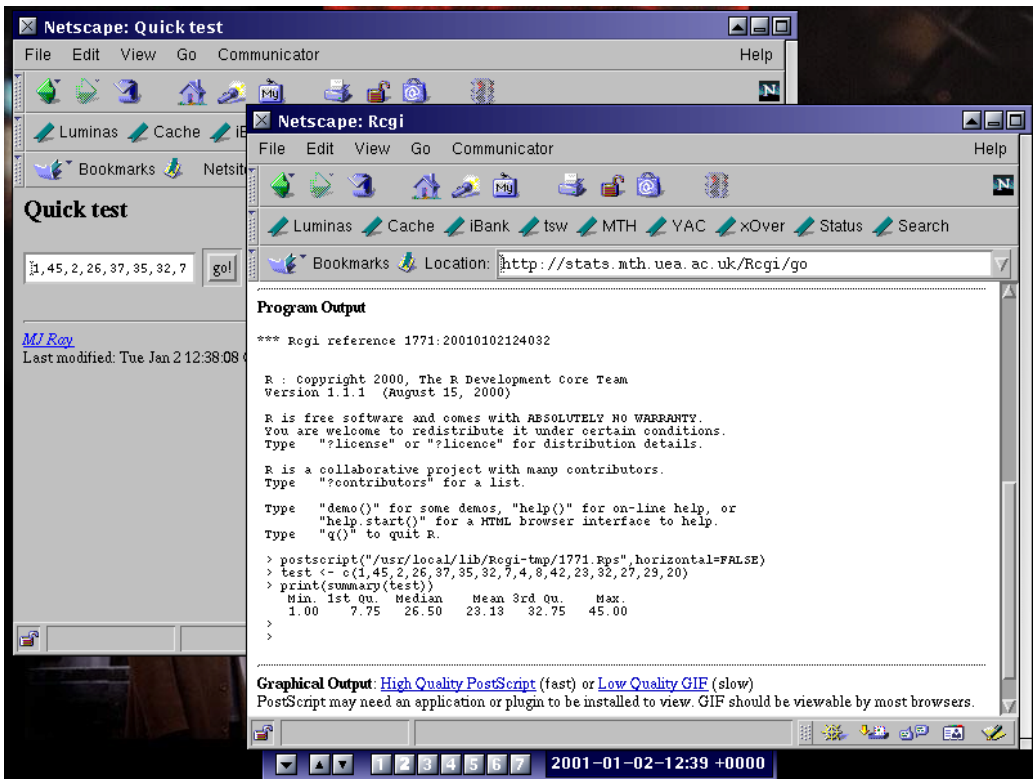

Figure 4: Screenshots of the results of the example HTML

programs it requires and build a module called 'Rcgi::SystemPrograms' detailing the location of some system programs required. The installer is still advised to review the scripts to make sure the details found are correct, though.

Other forthcoming improvements include better security checking (which is fairly basic at the moment) and better load balancing and session management. However, it is already a useful and popular system with a growing user community, with a mailing list available for help and development. The download site is http://stats.mth.uea.ac.uk/Rcgi/.

M.J. Ray
University of East Anglia, Norwich, UK
mjr@stats.mth.uea.ac.uk

Omegahat Packages for R

by John M. Chambers and Duncan Temple Lang

Overview

As part of the Omegahat project (http://www.omegahat.org/), we are developing a collection of packages to support new directions in programming in the S language, as implemented in R (http://www.r-project.org/) or in S-Plus (http://www.insightful.com/products/splus/). The packages we describe here illustrate ways to communicate between R and other languages and applications. You may find the packages useful if you want to access code from R that is written in another language, or call R functions from another programming language or application. This includes the notion of embedding R in such a package, or vice versa.

We also comment on future approaches to writing such software, with particular reference to making it available in both R and S-Plus.

We start with a very brief description of each of the available packages. More details can be found at the Omegahat web site (http://www.omegahat.org/). You can find a slightly lengthier description of how the packages relate to each other and the current tools in the sections below. We also plan to describe individual packages in future editions of the R newsletter. Please note that all of these packages are work-in-progress, at various different stages of evolution. We would be very appreciative of any com-

ments on any of the topics.

The packages

There are currently five packages that provide different ways to communicate between R/S-Plus and other languages and applications:

XML facilities for reading and writing XML documents in S, allowing easier data exchange between S and other applications;

Java an interface for calling software written in Java from R and R functions from Java, and includes an R graphics device which uses the Java graphics facilities and can be customized using S functions;

RSPerl an interface for running Perl inside R, and R inside Perl so that one can create Perl objects, and invoke their methods and Perl subroutines from R, and vice-versa;

Python like the Perl and Java interfaces, this allows R functions to be called from Python scripts and Python classes, methods and functions to be called from R;

CORBA dynamic facilities for invoking methods in other applications, on other machines and written in a different language and also providing these types of server objects in S itself. In other words, this provides a high-level mechanism for distributed computing in S.

Inter-system interfaces

While S is a general programming language with which one *can* do almost anything, it is not always the most appropriate tool. Most of us will have encountered situations in which it is better to use other software, whether it be motivated by the need for efficiency or just accessing functionality already written in another language. The interfaces from S to subroutines in C or FORTRAN and to shell commands have been around a long time and provide the basic mechanisms to access other software. The new interfaces provide access to new languages. They also provide *better* interfaces, both in terms of how they communicate and also in the ability to embed the S language software in other systems. Embedding is important if we want to bring statistical software to the users of non-statistical systems.

The inter-system interfaces provide three basic ways to communicate between R or S-Plus and other applications. The first is to share data in a common format. We suggest using XML, the eXtensible Markup Language to do this. The next approach is to embed the application within R or vice-versa allowing direct communication within a single process housing the two systems. The **Java**, **RSPerl** and **Python** packages do this for each of these different languages. The last approach is to support inter-process, inter-machine communication and for this we use CORBA.

XML provides a way of specifying data so that it is self-describing and can be read by different applications from a single source. XML provides a way of saying what gets transferred between the applications, not how, be it files, URLs, sockets, databases, etc. The inter-language and CORBA interfaces provide the mechanism of how data is transferred, but focuses more on the exchange of functionality between the systems. We won't make further mention of the XML approach here as it is described in another article in this newsletter.

Perl, Python and Java

The Perl, Python and Java interfaces allow one to create objects, call functions and methods, and evaluate expressions in these languages as if they were local to R. A module or package written in any of these languages can be installed and loaded into the session without having to write any special code. Thus, you can use Perl's modules to access network services, Web facilities, operating system functions, etc. Similarly, one can use any of Python's rich set of modules and any package implemented in Java, including building graphical interfaces, communicating with databases, etc. Additionally, from R, we can even find out what modules, classes, routines and methods are available to us from these different systems and how to call them.

An immediate consequence of these interfaces is that one can replace calls to scripts in each of these languages with a direct call to evaluate an expression in Java, Python, or Perl. In other words, rather than calling a script written in any of these languages via the `system` function, we can call the language directly within the R session and have it execute that script. For example, we can use `.PerlExpr` to evaluate a Perl expression or `.PythonEvalFile` to execute a Python script.

Direct calls have some key advantages: we avoid starting a new process each time; results from earlier evaluations can be used in future evaluations; and objects are returned rather than lines of text containing these values. In fact, we can return *references* to objects, meaning that, say, the Java object can stay in Java rather than being converted and copied to R.

Let's consider a simple example. Using Java/Omegahat embedded inside R, we create a new window containing a button. We can then do some additional computations in R and return to Java to access the button and set the text it displays.

```
.OmegahatExpression("f = new
  GenericFrame(b = new JButton('No text'))")
  ...  # S code
.OmegahatExpression("b.setText('Get help')")
```

While using strings to send instructions to the other language may appear convenient at first, it is clumsy, error-prone and inefficient. Instead, it is better to invoke routines and methods directly from within the S language, passing the arguments as regular S objects, avoiding the awkward and limiting pasting of the arguments together to form the expression as a string. For example, we can create a network news reader, set the news group to read and get the first article with the S commands

```
news <- .PerlNew("News::NNTPClient")
msgNums <- news$group("comp.lang.python")[[1]]
news$article(as.integer(msgNums[[1]]))
```

The R-Perl interface handles invoking the corresponding methods in the NNTP object in Perl and converting both the arguments and the results to and from Perl.

This approach makes these foreign routines appear as if they are local S functions and leads to richer and more flexible results. The "magic" behind the interface is that we can refer to objects in these other languages directly as if they are local to S, and the interface handles the references appropriately. Clearly, in the Java example above, there is no sensible way to copy the button or the window/frame to an R object. In fact, we want to leave it in Java and manipulate it in subsequent R calls, such as

```
.Java(b, "setText", "Get Help")
```

Each of these interfaces also provides a mechanism for calling S code from that other language. This means that we can call code in that other language and have it callback to R during the execution of that code. Also, it means that users of those languages can access sophisticated statistical software in a way that is familiar to them without having to learn yet another language.

Inter-process communication: CORBA

The approach of embedding other applications and languages within the S session does not work easily for all systems. For example, sometimes we will want to run these other systems on a different machine. So, we need a way to access functionality in other processes to things such as retrieving or updating the data in a spreadsheet; allowing computations to be displayed in a GUI, etc.

The **CORBA** package (http://www.omegahat.org/RSCORBA/) provides this functionality, allowing users to call methods in remote objects as if they were local S functions. The remote objects can potentially be in a different application, running on a different machine and developed in a different programming language. Also, S users can offer facilities to other applications by creating CORBA servers using S objects and functions. They do not need to fuss with any of the details of using CORBA with C/C++.

In addition to being able to share functionality from other applications, the **CORBA** package provides a simple, high-level way to implement efficient and structured (i.e. fault tolerant and synchronized) parallel/distributed computing. All one need do is have different R processes running on different machines, each of which provides a CORBA server to perform part of bigger task. Then one R process acting as the manager calls these methods in the different servers as background or asynchronous tasks which run concurrently.

In addition to these interfaces, we are also working on embedding R in the Postgres database management system, the Apache Web Server and within Netscape.

The S language

The S language, originally developed at Bell Labs, has become a widely used medium for doing data analysis and for implementing the results of statistics research. Its influence was recognized in the 1998 ACM Software System award.

What about the future of S? What steps can we take that will increase its usefulness and relieve some of the current difficulties?

Compatibility between R and S-Plus

We regard as a major goal to define and support a growing compatibility at the level of an Application Programmer Interface between R and S-Plus. Much of the software described in this article can run in both R and S-Plus. This means that users of these and other new packages can take advantage of other facilities in both languages, or work on platforms that are only supported by one of the implementations.

There are many ways to make new software available in both implementations of the S language. Other things being equal, one would like the differences in the implementation to be minimized. To do this, one needs software and guidelines that implement a subset of the S language compatibly for both systems. As time goes by, we want that subset to grow.

A natural side-effect of such an approach is a possible "standard" for the S language. An agreement among users and those responsible for S-Plus and R on a standard application programming interface for the S language would enhance the value of the language. Potential users and programmers would be encouraged that software written against such a

standard would be widely usable, just as such assurances are important for those programming in languages, such as C and C++, for which an agreed standard is available. Ongoing work in the Omegahat project will, we hope, encourage discussion of such a standard.

Summary

We have described very briefly some ongoing work, part of the Omegahat software, that provides facilities to programmers in R and S-Plus. Using these, you can connect software in the S language in an effective way to many other computing tools. Wherever possible, the resulting software should be directly usable in both R and S-Plus.

We encourage readers to get more information from the Omegahat website (http://www.omegahat.org/), and to try out the components that seem interesting for your applications. Because Omegahat is a joint, open-source project, we also particularly encourage suggestions and contributions to any of these efforts.

John M. Chambers
Bell Labs, Murray Hill, NJ, U.S.A.
jmc@research.bell-labs.com

Duncan Temple Lang
Bell Labs, Murray Hill, NJ, U.S.A.
duncan@research.bell-labs.com

Using XML for Statistics: The XML Package

by Duncan Temple Lang

XML, the eXtensible Markup Language is fast becoming the *next big thing* in computer applications and the Web. Along with the usual hype, there really is some substance and XML will probably turn out to have non-trivial relevance for statistics. In anticipation of this, we in the Omegahat project (http://www.omegahat.org/) have added an S package (i.e., works with both R and S-Plus) to both read and write XML. In this short article, we will outline some of the problems for which XML is a natural solution, and then describe what XML is, and some other areas in which XML technology is being exploited. Then we will take a brief look at some of the aspects of the R/S-Plus **XML** package. More information on the functions can be found at http://www.omegahat.org/RSXML/.

Using XML for datasets

Many of us are familiar with receiving the contents or values of a dataset in one file and a description of the dataset and its format in another file. Another problem is when the dataset is given as a set of files where records in one file correspond to records in the other files. In either case, the user must figure out the exact format and appropriate commands to group the values so as to read the values into into the data analysis environment. For example, the user must know whether some of the (initial) lines are comments; whether the first row of data contains variable names or is an actual record; how are missing values represented; whether integer values in a column represent a factor, a real valued-variable, or actually an integer; etc. The separation of the auxiliary information such as the ones just listed and other details such the author, the version of the dataset, the contact address, etc. can make it harder to process and ensure that the data has been read correctly.

When the datasets are not simply tables of numbers or strings, things quickly get more complicated. For example, if each record contains a different number of observations for a particular variable (i.e., ragged rows), some representation must be encoded into the ASCII file indicating which values are associated with which variable. Similarly, if the data frame of interest contains actual S objects that are not merely simple scalars, but are made up of a collection of values (e.g., the first two values of a record define a time interval object), we need to process these in a separate step after reading all the values using `read.table`.

If we could provide structure and additional information to data, we could also combine different, related datasets. For example, we can have both data about computers in a network and also the topology of that network in different parts of the same file. Also, we could add auxiliary information such as the number of records, the number of elements in a list, etc. This makes it easier and more efficient to read into S.

In order to address these issues, we need to find a way to add this type of structural information to the values. To do this, we need to agree on a format and then develop tools to read and write data using this format. And we need to develop these tools for different systems such as R, S-Plus, Matlab, SAS, etc. Well this seems like a lot of work, so we should look for an existing format that allows us to add our own