

No need to talk to strangers

Cooperation of Interactive Software with R as Moderator

Simon Urbanek

Simon.Urbanek@math.uni-augsburg.de

Department of Computer Oriented Statistics and Data Analysis,
University of Augsburg, Germany

Interface 2002

Abstract

Interactive software is very helpful for exploratory analysis of data, but most packages are optimized only for a specific field of work. Interaction between the various projects is limited. On the other hand very flexible statistical software, such as R, S or S-plus exist covering the needs of almost all fields, but without support for interactive graphics. This paper describes our experience with different approaches linking R with interactive software using flat files and the Omegahat interface. Such linking directly extends capabilities of the interactive software without the need for special communication layers such as CORBA. We also show how such an interface can be used to implement cooperation between different interactive software packages with R as moderator. Practical examples are given based on our current projects Klimt and Mondrian.

1 Introduction

Interactive software is very helpful for analyzing data. It allows us to see structures in the data and explore the data. Interactive features like multiple views, zoom, pan, linked highlighting and hot selection are very valuable tools for data analysis. The main goal is to provide an user-friendly interface with many options of visualization with a consistent look and feel.

In order to maintain consistency most interactive programs concentrate on a specific field or task. They are not extensible, additional functionality cannot be added. Interactive solutions are mostly stand-alone and isolated. More sophisticated interfaces to other software are scarce, usually consisting of plain text files for importing datasets and reporting results.

On the other hand, very flexible and extensible statistical environments such as S, S-plus or R have been developed. Anyone familiar with the environment can contribute new functionality and improve existing methods. Many interfaces to different languages and packages exist.

One of the things that these statistical environments do not provide is interactivity. The limited graphical functionality driven by a command line interface is suitable for use in publications, but is less practical for data analysis. Also the environment is too complex for performing simple tasks. Opening

a dataset and having a quick look at the data takes seconds in an interactive environment, but requires several lines of code in a statistical environment.

Since the two approaches complement each other well, the question arises whether it is possible to combine them and if so, how.

The motivation in our case was the cooperation of R and our interactive software for visualization and analysis of trees - KLIMT (KLassification - Interactive Methods for Trees) - Urbanek and Unwin (2001). Tree models can be grown in R using the available tree generators with all the flexibility of R. Then such a model is visualized and analyzed in KLIMT with feedback to the model construction in R.

We have tried several interfaces for this purpose which are described in the second section. Since this allows us to link multiple programs together, the extension of this idea to cooperation between interactive software packages and the possible setups are described in the third section. The methodology is illustrated using communication between our interactive packages KLIMT and Mondrian via R and $\hat{\Omega}$.

2 Interfaces

We have used two different interfaces to communicate between R and KLIMT: flat files (FF) and the Omegahat interface ($\hat{\Omega}$).

FF is the easiest to implement. It is a master/slave setup where ASCII text files are used to pass data between the applications. The calling application (*master*) creates the input file and launches the *slave* application as a new task. The master suspends its execution until the slave processed the input file and created an output file with the desired results. Here the usage is illustrated on the R/KLIMT example.

With KLIMT used for visualization and R for model generation two different setups are possible. Either R is chosen as master and calls KLIMT for visualization, or KLIMT as master calls R for tree generation.

When R is used as master, it needs to pass the dataset and the tree model to KLIMT. Both are written to a single text file, which is then parsed by KLIMT. In order to make the call more user-friendly it can be incorporated in an R-function as follows:

```
Klimt <- function(t,d) {
  write.table(d,"klimt.tree"); # create klimt.tree file
                                # containing the dataset
  sink("klimt.tree",TRUE);     # append further output
  print(t);                    # append the tree itself
  print(formula(terms(t)));    # and finally the formula
  sink();                      # close the output file
  system("klimt klimt.tree");  # call KLIMT passing the
};                             # filename as argument
```

Since no special conversion routines are used, KLIMT needs to be able to read the default formats of R. For more complex data to be passed between the applications, corresponding output routines and parsers have to be written at both ends.

Let's illustrate the typical call to the `Klimt` function using the *iris* dataset which is provided with the default R package. We need to load the dataset, construct a tree which takes *Species* as the response variable and all other variables as predictors. Finally the `Klimt` function is used to start KLIMT passing the tree and the dataset:

```
data(iris);
Klimt(tree(Species~.,iris),iris);
```

R is suspended until KLIMT terminates. We don't process any parameters passed back from KLIMT, which could be done by using another output file or using the `intern=TRUE` parameter of the `system` function. Parallel execution is possible on some platforms, but the behavior is system-specific.

An alternative way of using the FF interface is to use KLIMT as master application which calls R to construct a tree model. R provides so-called *batch mode* to simplify this approach. In this mode R reads an input file containing commands and produces an output file with results. The command file is generated on-the-fly according to the interactive selections made by the user. This way the user gets a fully intuitive graphical frontend to the tree construction functionality of R. Beside the command file, KLIMT needs to pass the dataset as well. This is done by storing the dataset in a separate file. Since R in batch mode creates an output file automatically, the `sink` function doesn't have to be used.

The FF interface is very simple and requires no further setup or external libraries, but it has still several drawbacks. Unfortunately calling conventions are platform dependent. There is no `R.exe` for windows, and the provided `Rterm.exe` lacks support for the batch mode. Moreover R is by default not in the `PATH` making it impossible on the Windows platform to automatically detect the current R installation. This shortcoming can be circumvented by providing your own scripts and wrappers, which counteracts the simplicity of the FF.

A further disadvantage is the lack of monitoring, which is frustrating for the user especially if the required task is computationally intensive, as the user doesn't know whether the machine has crashed or is just too busy solving the problem.

Finally there is no direct communication between the applications. The processes cannot influence each other, no callbacks are possible. It is an all-or-nothing situation.

An appealing alternative to solve these problems is provided by the *Omega-hat* group. They concentrate on developing native interfaces for the R/S/S-plus family of statistical environments to many different languages and development platforms. Since most of our projects, including KLIMT are Java-based, we want to concentrate on the SJava interface of $\hat{\Omega}$. SJava allows the calling of R-functions from Java and vice-versa, making it suitable for our application.

A sample usage of $\hat{\Omega}$ for calling Java software from R is illustrated by the following code fragment:

```
library(SJava); # load SJava
.javaInit(list(classPath=c("klimt.jar"))); # initialize JVM
k<-.JavaConstructor("Klimt","data.klimt"); # start KLIMT
.java(k,"addTree","tree.klimt"); # add a tree
```

The main advantage over FF is that both R and the interactive application run in parallel and can cooperate. The call to `.JavaConstructor` returns an object which can be used for any further calls, such as adding new trees, selecting cases and so on. We use this method for construction of forests, where a function in R constructs bootstrapped trees which are then passed to KLIMT for further analysis, while work on previously loaded trees can be done in parallel.

It is also possible to create callback functions in R which are called in response to events in KLIMT, such as selecting cases or the interactive changing of parameters. More detailed information about these techniques was presented by Temple Lang and Swayne (2001). We don't want to discuss technical details here, such as object type conversion, since nice wrappers can be created to hide all this "magic" from the user as the following example illustrates:

```
source("klimt.r");           # load wrappers
data(iris)                  # get dataset
k<-Klimt(tree(Species~.,iris),iris) # start KLIMT
k.tree<-Klimt.getTree(k,0)  # get 1st tree object
k.mark<-Ktree.getMarker(k.tree) # get marker
Kmark.clear(k.mark);       # clear selection
for(i in 0:49)              # select first 50 cases
  Kmark.select(k.mark,i)
Kmark.update(k.mark);      # update all plots
```

In this example a tree is generated, KLIMT is started and the first 50 cases are selected in all plots. In a similar way new plots can be opened or their parameters can be modified.

Analogously Java programs can call R to perform statistical computations. In this case parallel processing doesn't play as important a role as in the other direction, where the event loop of the Java program runs in a separate thread. Nevertheless considerable speed up can be achieved, since only one instance of R needs to be used and objects don't have to be re-loaded for each new computation, as opposed to the FF approach.

Although $\hat{\Omega}$ provides a more flexible and versatile interface than FF, its main drawback is the fact that it has to be setup before usage. A FF interface can be used in most cases regardless of the configuration, making the deployment of applications very easy. $\hat{\Omega}$ is still an experimental interface with main support for unix platforms without any automated setup procedure for other platforms. Still, the more projects use $\hat{\Omega}$, the better the support is likely to be.

3 Cooperation

As we have seen $\hat{\Omega}$ provides very nice features for combining R and interactive programs. There are already several programs using this technology, such as ggobi, ORCA, KLIMT and Mondrian. This lead us to the idea of using R as a moderator for communication between the different interactive programs, without the need of additional layers, according to the motto "Don't talk to strangers, talk to R". In our case the motivation was to use KLIMT as an extension of Mondrian to support classification and regression trees.

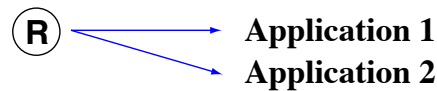
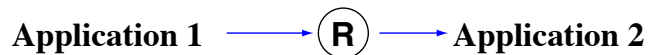
Setup 1: Controlled by R**Setup 2: Controlled by application**

Figure 1: Different types of cooperation setups.

There are two possible cooperation setups. In the first setup R manages the entire communication. The second setup requires one application to launch an instance of R and to use it for startup and communication with the other application, as illustrated in fig. 1.

The R-controlled setup doesn't require any application to know about the others. The only requirement is the support for some very basic API, e.g. to register a new menu item or access selected variables. Let's say we want to add a new menu item *Generate tree* in Mondrian which should cause R to generate a tree from the selected variables and start KLIMT. Then another menu item in KLIMT should make Mondrian select exactly the same cases as selected in KLIMT.

In an R controlled setup all the work is done by an initializing R function setting up the cooperation and a callback function handling responses from menu items. All we need from each application is some way to get their `MenuBar` object to be able to add a menu item. If they are already derived from the `Frame` class, then a simple `mb<- .Java(app, "getMenuBar")` is all we need to obtain it. An instance of `ActionListener` can be generated on-the-fly in R and associated with the corresponding callback function to handle events triggered by both newly added menu items. Starting each application, loading trees in KLIMT and modifying selection has been already shown in the examples above. It is even possible to build a modal dialog for parameter selection entirely in R. This means that in order to achieve the desired functionality none of the applications has to be modified in any way. The setup is fully symmetric with respect to the applications. It also allows linking of multiple different interactive packages.

The application controlled setup is asymmetric, because the master application initializes both R and the slave application. The master application first starts an instance of R and issues R-commands to start another application. In this setup the backflow of information is more limited, and rather cumbersome, since two layers are crossed. Where in the previous setup every callback just went from $JVM \rightarrow R$, in this setup it must pass $JVM_1 \rightarrow R \rightarrow JVM_2$.

	Java from R			R from Java		
	OS X	Linux	Win	OS X	Linux	Win
available	✓	✓	✓	✓	✓	✓
ext. libraries	✓	✓	✓	✗	✓	✓
stable	✓	✗	✓	○	○	○

Tested with R-1.4.1, SJava 0.64-0 and Sun JDK 1.3.1

MacOS X: our own port of SJava 0.64-0; native R-1.4.1 for MacOS X installed from fink package

Windows: SJava compiled from sources with MinGW; both JDK 1.3.1 and 1.4.1 were tested

Legend: ✓ yes ✗ no ○ yes, if certain criteria is met

Figure 2: Compatibility chart based on our tests.

A hybrid approach where the master application constructs R-code which handles the communication is also possible, but hard to implement and debug.

Although less flexible than the R-controlled setup, the main purpose of this setup is to hide the overhead necessary for the interconnection of both applications from the user. The user still sees only the interactive frontend and doesn't have to care about callbacks or commands. We have implemented application controlled setup with Mondrian as master, calling R for tree construction and finally launching KLIMT in R via SJava.

4 Stability and Availability

The $\hat{\Omega}$ project is very useful, but it is still work in progress. We have tested the SJava interface on various platforms and the result summary is shown in fig. 2. Both ways were tested: calling the R interpreter from Java and creating Java objects from R. We chose three main platforms for our comparison: MacOS X, Linux and MS-Windows. Linux is used as a typical unix-based system, MacOS X is used on most Apple Macintosh computers and is also unix-based, but with some differences that prevent the use of out-of-the-box packages. Since there is no official release for MacOS X we have used our own port of SJava. Finally we chose Windows as the most popular platform in the non-research world, in particular we used Windows 98, 2000 and XP. There is only a fairly old release of SJava for Windows available on the internet so we decided to use the latest version compiled from sources. Our binary packages for MacOS X and Windows are available on request.

The main criteria were the availability for each platform, the possibility to use external libraries and finally stability for everyday use. SJava is available on all three platforms, for Linux as source code, for MacOS X as a binary package, both easily installable via `R CMD INSTALL`. Windows version is available as packed binary. Sun's JDK or at least JRE has to be installed and properly configured for SJava to work.

Calling Java from R turned out to work flawlessly on Windows and MacOS X. During the test both versions of the interface were stable. On the Linux platform there are some known stability issues. It means that the interface works, but sometimes random, unexpected crashes occur. This problem is being addressed and should be solved in next releases of R.

We were able to successfully test SJava on all platforms when used the other way: calling the R interpreter from Java code. Due to the special handling of shared libraries and the JNI framework in MacOS X, it is currently not possible to use external libraries that require native code in our ported SJava. Both other platforms supported the use of external libraries. It is even possible to load the SJava library in this setup, which suggested the idea of application controlled cooperation as described above.

Unfortunately our test showed some deficiencies in the interface. There were deterministic crashes, caused by certain commands and expressions in R. Unlike the threading issue mentioned before, these crashes were reproducible and can be avoided by simply using alternative expressions to perform the same task. Interestingly the crashes were consistent across platforms and even versions of SJava. Because of this behavior we cannot classify this interface as stable, but it is usable since the errant commands can be avoided.

We came to the conclusion that the SJava interface is very useful with huge potential and once the mentioned flaws are eliminated, it can be used by many Java programs for communication with R¹.

5 Conclusion

We have shown that interactive software and statistical environments although being totally different by nature complement each other well. Two interfaces have been considered for communication between R and interactive programs written in Java: flat files without inter-process communication and the $\hat{\Omega}$ interface for flexible embedding.

The usage of flat files where generators and parsers on each side take care of the data flow turns out to be easier to setup. No parallel execution is possible, but also no third-party software is necessary. FF can be used when no other interface is available or for tasks that don't require much time. An alternative is provided by the $\hat{\Omega}$ interface, allowing parallel execution, use of callbacks and sharing of objects. There are already several applications available using $\hat{\Omega}$, which allows the initiation of cooperation between different interactive packages.

This communication can either be controlled by R in a symmetric setup, or controlled by the initial application in a master/slave setup. The R-controlled setup needs no modification of the participating application, but requires corresponding code in R, making this a very flexible option, but requiring knowledge of $\hat{\Omega}$ and the applications. The application-controlled setup is less flexible, but allows the integration of the control into the interactive software, hiding the internals from the end-user.

¹All our test were performed with R, but SJava is reported to work also in S/S-plus environments. We didn't have the opportunity to test such setup in practice.

The $\hat{\Omega}$ interface is very flexible and promising, but it is still a project under development. The available functionality is platform-dependent. There are stability issues on some platforms. Some flaws are deterministic, some are apparently random. Except for unix variations there is no automated setup procedure.

Seeing the many possibilities of the $\hat{\Omega}$ interface for interaction between statistical environments and interactive software, it remains to hope that the interface becomes more and more stable as new applications emerge, so our software doesn't have to talk to strangers.

References

Temple Lang, D. and Swayne, D. F. (2001), "ggobi meets R: an extensible environment for interactive dynamic data visualization," in *Proc. of the 2nd Int. workshop on distributed Statistical Computing, TU Vienna*.

Urbanek, S. and Unwin, A. R. (2001), "Making Trees Interactive - KLIMT," in *Proc. of the 33th Symposium of the Interface of Computing Science and Statistics*.

Links of interest

<http://www.omegahat.org/>

Omegahat project

<http://www.r-project.org/>

R-project

<http://www.klimt-project.com/>

KLIMT project

<http://stats.math.uni-augsburg.de/>

Pages of our department with links to KLIMT, Mondrian and other interactive software.