When to use the Tcl/Tk and Java packages?

Duncan Temple Lang

December 13, 2005

R now has packages that provide an interface to each of Tck/Tk and Java. With the existence of these two packages, each programmer has to decide which to use for a particular problem. In this document, we discuss some of the issues and hopefully provide some aid to deciding which toolkit to use for different types of projects.

A thing to keep in the back of one's mind when reading this is Einstein's remark

> a simple as possible, but no simpler.

Basically, while a simple approach to programming is desirable, if that approach prohibits or makes excessively difficult certain tasks, the simplicity is too simple.

1 General

First some general remarks. Tcl/Tk is a scripting language while Java is a programming language. Omegahat and other systems (e.g. the BeanShell, etc.) provide interactive access to Java's classes. One implication of this difference between the languages is that Tcl/Tk is good for writing short, simple applets that are unlikely to be modified or enhanced in the future. In other words, "programs" written in Tcl/Tk are more "write-once" code. In many respects, this is like Perl - a very rapid development tool for programming specific tasks, but providing little or no support for development of large and incrementally developed applications.

Tcl/Tk provides very sensible and commonly used default values which leads to an efficiency in writing code. Java provides much greater capabilities than the tools/"classes" in Tcl and Tk but at the expense of having to specify what one means. Additionally, the enhanced capabilities provided by the Java classes make it appear more complex than other less rich toolkits. For the novice, such complexity is overwhelming and can lead to the users preferring the simpler tool. While this is a very understandable reaction and a real issue, it is usually a short sighted gamble or compromise. If one needs and continues to need only the functionality that Tcl/Tk provides, then the simplicity is an added advantage. On the other hand, if the design of an application evolves over time (be it hours or months) and new capbilities are needed that the programming tools do not provide, one must translate the initial work to a different toolkit (e.g. Java).

The ease of use of Tcl and the Tk widgets has the effect of reducing the number of lines of code (i.e. commands). This can only be a good thing and is a significant advantage of the Tcl/Tk interface over Java and other interfaces. One can reduce this overhead by defining functions in R that provide many of these defaults. Additionally, one can develop high-level Java classes that extend the commonly used Java classes and provide these defaults.

In the examples we have constructed using the *.Java()* mechanism, the added complexity of Java relative the corresponding Tcl/Tk implementation is very slight. The code follows the same structure and is only fractionally lengthier.

2 Strings

A major difference between Tcl and Java is that Tcl relies heavily on strings. For example, numbers are represented as strings. While this does have performance drawbacks, this is not our concern here. Instead, we think this simplified interface is just too simple. Firstly, creating strings to pass to Tcl/Tk can sometimes be

complicated as it involves inserting the values of complex R objects into the Tcl expression. This can be non-trivial. Secondly, and more importantly, strings do not allow us to adequately express non-trivial semantic information. R and Java objects contain much more information than their textual versions. Finally, a string-based interface removes (or makes very difficult) a pass-by-reference semantic.

In many simple uses of the Tcl/Tk toolkit, these constraints do not introduce undue problems. In these cases, Tcl/Tk is fine. However, as an application becomes more complex, workarounds for these constraints make programming more complex and often "unnatural".

3 Classes and Reuse

Standard Tcl/Tk does not offer object-oriented facilities. One can write functions in the same way one defines routines in C. We believe that an object-oriented approach is the most valuable way to develop statistical software. Classes in Java allow code to be easily reused by inheritance/extension. Thus when developing Java classes, one can potentially use them in the current and *other, future* applications.

In addition to the usual benefits of the object-oriented approach such as inheritance, polymorphic methods, etc., non-trivial programs also benefit from the addition of an extra layer of name spaces - namely the scoping of fields within an object. As different elements are gathered together in a singl application, it is essential that they do not interfere with each other in subtle ways. In S and R, this often happens when programmers choose to implement a feature by assigning values to global variables. If two functions write to the same variable, chaos can ensue.

Such situations of conflicting use of global variables arise quite commonly in the type of environment provided by the R-Tcl/Tk interface. The nature of R allows a user to essentially run multiple small GUI applications within a single session and do this *concurrently*. The tkdensity example in the tcltk package in R illustrates this potential. Consider that we want to run two instances of this interface. Since both use the global `kernel` variable in the Tcl space (i.e. `\check@icr` tclvar$kernel), there will be contention between the code. Selecting a kernel in on Of course, one can program the example differently, but the issue of whether the environment supports good programming or makes it more difficult. If one insists that only one application is running at a time, the programming model is simple, and so is the result.

Classes and objects reduce the need to use global variables. Instead, objects contain their own version of the data they need to share across different "function" calls. This greatly improves program development and maintenance and code reuse.

4 Support – Documentation and Libraries

One of the reasons we believe Java is a promosing framework on which to build Omegahat and generally components for statistical computing is its widespread support in different communities. We believe that more tools, packages and classes will be available for it than most other languages and that we will be readily able to intergrate these into our applications. This has already been evidenced by the large number of packages released by Sun (JHelp, JTAPI, Java 2D and 3D graphics, etc.) and also by other vendors and projects (antlr, etc.)

Of major importance for the usability of a package is access to documentation for users at different levels of expertise. When travelling recently in Berkeley, CA and also Geneva, Switzerland I had the opportunity to contrast the availability of books on Java and Tcl/Tk in two excellent bookstores - Cody's and ?. In Cody's (CA), I found 5 books on Tcl/Tk. In Geneva, I found 3. For Java, each store stocked 12 rows of books, ranging in topic from algorithms, performance, GUI development, multi-threading, etc. In other words, there is a very rich literature for Java, including very advanced books and those for beginners.

5 R and S Compatability

For a long time, the developers of R have attempted to be as consistent as possible with S as possible and as made sense. Since the Tcl/Tk package is not available in S and is unlikely to be (given tha Splus offers a different mechanism for creating GUIs), there is now an incompatability between the two systems. For users

of R who might want to have their GUI code used in S in the future, the .Java() approach is the one that will allow this to happen. The Tcl/Tk package will not.

6 Conclusion

Tcl/Tk is fabulous for developing simple, short scripts. For more long-term, complex applications that have a long development and shelf-life, Java provides a rich, strongly-typed development environment and a large collection of classes for a myriad of different applications.

We believe that the combination of Omegahat & R to provide a familiar, high-level, interactive interface to Java's classes is a close to ideal compromise between the ease of use provided by scripting languages and strongly-typed, programming languages. It provides a smooth transition between programming in Java, Omegahat and R, allowing one to use the strengths of each language for different parts of an application.

Given that it is hard to remember the details of two GUI toolkits (e.g. the widgets and their properties, callbacks, etc.), most people will likely learn how to program in only one toolkit. If you know that you will only ever work with simple interfaces and will not need any of the advanced functionality of Java, learning Tcl/Tk seems like a sensible approach. However, if you want to keep your options open and be in a position to take advantage of an ever-increasing collection of integrated classes, Java is a significantly more flexible environment and only marginally more complex given the interactive access provided by Omegahat and the R and S Java packages.

An additional consideration in deciding which package to use is that the Java interface is in both Splus and R whereas the Tcl/Tk package is only available in R.