DSC 2003 Working Papers
(Draft Versions)

http://www.ci.tuwien.ac.at/Conferences/DSC-2003/

Rserve
A fast way to provide R functionality to applications

Simon Urbanek

Department of Computer Oriented Statistics and Data Analysis
University of Augsburg

Abstract

Rserve is a TCP/IP server which allows other programs to use facilities of R from various languages without the need to initialize R or link to R library. Every connection has a separate workspace and working directory. Client-side implementations are available for popular languages such as C/C++ and Java. Rserve supports remote connection, authentication and file transfer. This paper describes the Rserve concept, compares it with other techniques and illustrates its use on several practical examples.

1 Introduction

The R statistical environment provides a powerful suite of tools for statistical analysis for use in many scientific fields. Its application is not limited to statistical research only, but its modular design allows the use of R and its packages in other projects. Often it is feasible to hide the programming aspects of R from the user and integrate the computing capabilities of R into a customized software designed for a specific target group. Possible examples are web-based servlets that allow the user to analyze his/her data using a fixed process, or software for data visualization which uses R facilities to generate statistical models.

R provides two native interfaces for communication with other applications: a simple standard input/output model or R as a shared library. As we will describe later they are unsatisfactory in many situations, either because of speed concerns or when the host application is not written in the C language. In this paper we propose another method of using R facilities from other applications, called Rserve. It is not limited to specific programming languages and even allows separation of client and R environments. The main concerns while developing the system were speed and ease of use. In section 2 we describe the design of Rserve, in section 3 we compare Rserve to other communication methods, including the Omegahat(3)(1) approach. In section 4 we illustrate the use of Rserve on several basic examples. A real application of Rserve is described in section 5. Concluding remarks and ideas for the future are mentioned in section 6.

2 Design

Our experience with other communication methods has shown that there are three main points to be considered when designing a new system: separation, flexibility and speed. It is important to separate the R system from the application itself. One reason is to avoid any dependence on the programming language of the application, since a native direct interface to R is usable from the C language only(2)(5). Another aspect comes from the fact that tight integration with R is more error prone, since the application must take internals of R into account. On the other hand application developers want the interface to be very flexible and make use of most R facilities. Finally speed is a crucial element, because the goal is to provide the user with the desired results without the need of starting an R session from scratch.

A client/server concept allows us to meet all three key requirements. The computation is done by the Rserve core, which is a server answering requests from clients such as applications. The communication between the Rserve and the client is done via network sockets, usually over TCP/IP protocol, but other variations are also possible. This allows the use of a central Rserve from remote computers, but also local communication on a single machine.

One Rserve can serve multiple clients simultaneously[1]. Every connection to Rserve obtains its own data space and working directory. This means, that objects created by one connection don't affect other connections in any way. Additionally each connection can produce local files, such as images created by R's bitmap graphics device, without interfering with other connections. Every application can open multiple connections to process parallel tasks.

The data transfer between the application and Rserve is performed in a binary form to gain speed and minimize the amount of transferred data. Intermediate objects are stored in Rserve, therefore only objects of interest need to be transferred to the client. For practical examples, see section 4.

Beside communication with the R core, Rserve has also an integrated authentication and file transfer protocol, which makes Rserve suitable for use on separate machines. User authentication is provided to add a level of security for remote use. Currently Rserve support two variants: plain text authentication and unix crypt authentication with server challenge[2]. File transfer allows copying of files either needed for the computation or produced by R from the client to the server and vice versa. Both features can be turned off if Rserve is used only locally.

The Rserve protocol used for communication between Rserve and the client is synchronous, since R cannot handle multiple requests. The client sends a command to be processed, Rserve sends an answer as soon as the result becomes available and

[1]An exception to this rule is the Windows operating system, which lacks an important feature (*fork*ing of processes) that allows parallel execution of R instances. Windows version of Rserve can serve connections only sequentially and without namespace separation. Multiple Rserves or R-(D)COM may be better solution for applications that use parallel R processes.

[2]The server supplies the requested salt and the client must respond with the correctly encrypted password.

awaits next commands. Currently Rserve supports two main groups of commands for communication with R: Creation of objects in R and evaluation of R code.

Most basic objects, such as numbers, strings or vectors can be constructed via direct object creation. The contents of the objects are sent in binary form from the client to the server. This provides an efficient way of transporting data necessary for evaluation. All objects are always passed by value to separate client and server data spaces. This way both the client and the server are free to dispose of the data at any time, preventing crashes which are inherent in other communication methods where the systems share the same data physically.

The second main command group is the evaluation of R code. As opposed to object creation such code is sent in clear text to Rserve and is treated as if the code was typed on the console in R. The resulting object of the evaluation can be sent back in binary form to the client if requested. Most R types are supported, including scalar numbers, strings, vectors, lists (hence classes, data frames and so on), lexical objects etc. This allows Rserve to pass entire models back to the client. The client may decide to not receive any objects, which is useful while setting up intermediate objects in R which are not directly relevant to the client.

A typical use of Rserve facilities is to load all necessary data into R, perform computations according to the user input, such as construction of models, and send results back to the application for display. The data can be sent either directly as binary objects or as a data file, depending on the purpose. All data and objects are persistent until the connection is closed. This allows an application to open a connection early, pass all necessary data to the server and respond to user input by ad-hoc computing of the desired models or estimates. Since the results are not in textual form, no tedious parsing of the results is necessary.

The interface to Rserve is modular and documented, allowing access to Rserve from any application or programming language which supports sockets, including all current scripting and programming languages. We have implemented a client for Rserve in pure Java, which interfaces to most facilities of Rserve and maps all objects available in Rserve into native Java objects or classes. The use of the Java client is illustrated in the examples section.

3 Comparison with other methods

Rserve aims to complement the variety of available methods for communication to R, not to replace them. Native API for communication with R is defined on the level of the C or Fortran languages, excluding other languages unless some kind of a bridge is used, specific for each language. Rserve provides an interface which is defined independently of any programming or scripting language.

A console-based interface, where commands for R are stored in a file and written into another file (also known as batch mode) is currently used by several applications, such as VASMM(4). It is considerably slow, because a new instance of R has to be started for each request. Rserve has very little overhead for each new connection, because it doesn't need to initialize a new instance of R.

Results are usually stored in textual form, which is not suitable for interprocess communication and requires a parser at the application's end. This may not be a problem for some scripting languages such as perl, but it is a problem for other languages such as Java. Rserve still provides a way of capturing textual output if necessary, although the preferred method is binary transfer.

Since our main applications of Rserve involve the Java client for Rserve, we also compared Rserve to the SJava interface from the Omegahat project. SJava is conceptually far more flexible than Rserve, because it allows calling both R from Java and Java from R. Rserve implies that Java is the controlling application and unlike SJava it has no concept of callbacks. Every action is initiated on the Java side and performs computation in R.

Let us compare Rserve with the R-from-Java part of SJava, because they are based on very similar philosophies. Rserve can be used remotely, because objects are copied when necessary. This approach allows distributed computing on multiple machines and CPUs. SJava works only locally since it embeds R into Java via JNI. Due to the fact that there is no synchronization between Java and R, and given that R is not multi-thread safe, it is fatal to make more than one call from Java into R. The application developer is responsible for proper synchronization when using SJava. Rserve performs this synchronization by design and also allows the use of multiple concurrent connections. SJava allows passing of object references, which can lead to serious problems and crashes if utmost care is not taken.

Both SJava and Rserve support conversion of basic object types between Java and R. Rserve provides much a wider variety of objects passed to Java by encapsulating all native *SEXP* (simple expressions) of R into a Java class. In SJava conversion of complex types is supported, but the developer must implement his own class converters.

Probably one of the main disadvantages of SJava is that it does not run out-of-the-box. The code is dependent on the hardware and operating system used, as well as the Java implementation. It is in general hard to setup and the solution cannot be deployed with the application. Rserve comes as a regular R source package for unix platforms and as a binary executable for Windows. The client side of Rserve needs no special setting and is platform independent, since it is written in pure Java. The Rserve client classes can be easily deployed with any Java program and no third party software is necessary.

Finally R has its own set of functions for socket communication, therefore it should be possible to build a pure R program mimicking the same functionality as Rserve. Although this is true, such an R program would only be able to serve one connection at a time and would lack separate workspaces. The use of the serializable format of R instead of the Rserve protocol was also suggested, but the serialization is known only to R and therefore the application would have to implement the full serialization protocol. Only limited documentation was at our disposal when the decision was made, therefore we decided to use our own binary protocol.

4 Using Rserve

Rserve itself is provided as a regular R package and can be installed as such. The actual use is not performed by the `library` command, but by starting the Rserve executable (Windows) or typing `R CMD Rserve` on the command line (all others). By default Rserve runs in local mode with no enforced authentication. Once the Rserve is running any applications can use its services.

All of our applications using Rserve represent Java programs which use R for computation, therefore we will show examples using the Java client for Rserve. The principles are identical when using other Rserve clients, therefore using Java as the starting point poses no limitation.

Before plunging into real examples, let us consider the minimal “hello world” example:

```
Rconnection c = new Rconnection{};
REXP x = c.eval("R.version.string");
System.out.println(r.asString());
```

The code has the same effect as typing `R.version.string` in R. In the first line a connection to the local Rserve is established. Then the R command is evaluated and the result stored in a special object of the class `REXP`. This class encapsulates any objects received or sent to Rserve. If the type of the returned objects is known in advance, accessor methods can be called to obtain the Java object corresponding to the R value, in our case a regular `String`. Finally this string is printed on the standard output.

The following code fragment illustrates the use of slightly more complex native Java types:

```
double[] d = (double[]) c.eval("rnorm(100)").getContent();
```

The single line in Java provides an array of 100 doubles representing random numbers from the standard normal distribution. The numeric vector in R is automatically converted into `double[]` Java type. In cases where no native Java type exists, Rserve Java client defines its own classes such as `RList` or `RBool`[3]. This approach makes the use of Rserve very easy.

As a first more practical example we want to calculate a Lowess smoother through a given set of points. The Java application lets the user specify the data allowing interactive changes of the points, displays a regular scatter plot and needs coordinates of the smoother to be obtained from R.

One way of obtaining such a result would be to construct a long string command of the form lowes(c(0.2,0.4,...), c(2.5,4.8,...)) and using the `eval` method to obtain the result. This is somewhat clumsy, because the points usually already exist in a `double` array in the Java application and the command string must be constructed from these. An alternative involves constructing objects in R directly. The following code shows the full Lowess example:

```
double[] dataX,dataY;
...
Rconnection c = new Rconnection();
c.assign("x",dataX);
c.assign("y",dataY);
RList l = c.eval("lowess(x,y)").asList();
double[] lx = (double[]) l.at("x").getContent();
double[] ly = (double[]) l.at("y").getContent();
```

First the Java application defines the arrays for the data points *dataX* and *dataY*. The application is responsible for filling these arrays with the desired content. Then we assign the contents of these arrays to R variables x and y. The `assign` command transfers the contents in binary form to Rserve and assigns this content to the specified symbol. This is far more efficient than constructing a string representation of the content.

[3]Java’s boolean type has no support for NA missing values, therefore it cannot be used to directly represent logical type in R.

Once the variables are set in R we are ready to use the `lowess` function. It returns a list consisting of two vectors x and y which contain the smoother points. The `RList` object provides the method `at` for extraction of named entries of a list. Since lists may contain entries of different types, the object returned by the `at` method is of the class `REXP` whose content can be cast into `double[]` in our case. The result can now be used by the Java application.

More complex computations can be performed even without transmission of resulting objects. This is useful when defining functions or constructing complex models. Model objects are usually large, because they contain original data points, residuals and other meta data. Although they can be transferred to the client, it is more efficient to retain such objects in R and extract relevant information only. This can be done by using the `voidEval` method which does not transfer the result of the evaluation back to the client:

```
c.assign(y, ...) ...
c.voidEval("m<-lm(y~a+b+c)");
double [] coeff =
  (double[]) c.eval("coefficients(m)").getContent();
```

In the above example a linear model is fitted, but its content is not passed back to the client. It is stored in an object in R for later use. Finally the coefficients are extracted from the model and passed back to the Java application.

So far we used Rserve in local mode only. Extension to remote Rserve connections is possible without code changes, except for additional parameters to the `Rconnection` constructor, specifying the remote computer running the Rserve. For details about the use of remote authentication, error handling and file transfer, consult the documentation supplied with the Rserve and the Java client. The use is again straight-forward, since native Java facilities, such as input/output streams are used.

5 Example

In the following we want to describe a real-life application of Rserve. The example features Klimt(6), a software for visualization and analysis of trees and forests. Klimt is written entirely in Java and provides numerous interactive facilities for visualization of tree models and analysis of associated data. Klimt can be used as a stand-alone application, but it requires R for the construction of tree models. Therefore it needs a way of communicating with R to perform the necessary computations.

There are four tasks for which Klimt connects to a Rserve: initialization, construction of a tree from the open data set, construction of tree branches when the user interactively modifies a tree and finally construction of derived variables.

When initializing R by opening the Rserve connection, Klimt checks the version of R and loads the necessay libraries for tree construction - `tree` or `rpart` depending on the user's choice. Before the first tree is generated or the first variable is used, Klimt stores the entire data set in R by assigning each variable of the data set into R objects of the same name. For tree construction Klimt simply evaluates an R expression of the form: `"tree("+formula+","+parameters+")$frame"`. The resulting object contains a data frame which entirely describes the tree. This information is converted by Klimt into an internal representation of a tree. The formula is generated from the items selected by the user from a list. Optional parameters can be specified by the user.

It is recommended to wrap the evaluated expression in the `try` function. The resulting object is then either the requested tree, or a string containing the error message if the command was not successful. In Klimt the actual code looks like this:

```
REXP r=c.eval("try(tree("+formula+","+parameters+")$frame)");
if (r.getType()==REXP.XT_STRING) {
  String error=r.asString();
  ...
} else {
  SNode root=convertTree(r.asList());
  ...
}
```

Here `SNode` is the internal recursive representation of a tree in Klimt. A similar approach is used for interactive tree splitting. The user interactively specifies the split, resulting in two nodes. Two subsets corresponding to the interactively created nodes are used, one tree is grown for each node and attached to its parent node. The main advantage is that the connection is held open and therefore the data set doesn't need to be re-transmitted to Rserve.

Finally derived variables can be created by evaluating an expression supplied by the user and stored in the requested variable:
`REXP r=c.eval("try("+varName+" <- "+expr+")");`
If the expression supplied by the user is correct, then the result must be an array of the same length as the data set in Klimt. Since the variables are stored directly in R, expressions of the form `v1/v2+v3` deliver the expected result if the data set contains the variables $v1$, $v2$ and $v3$.

6 Conclusions

Rserve complements the family of interfaces between applications and R by providing a fast, language-independent and remote-capable way of using all facilities of R from other programs. Due to a clean separation between R (server) and the application (client), internal data manipulation on one side cannot affect the other. Using network sockets for the communication ensures platform and software independence of the client and the server. At the same time restriction to local use is also possible, requiring no physical network.

For concurrent connections Rserve offers both data and file space separation between connections. Each new connection is accepted almost immediately without the need for initializing the R engine. Integrated file transfer protocol allows the use of remotely created files, such as plot bitmaps created by R. User authentication is provided for some a level of security, especially when used in remote mode. This concept is suitable for distributed computing and load balancing.

The supplied Java client provides an easy embedding of R facilities into Java programs. Evaluation and transfer of most types from R to the application is provided, including complex objects such as models. All basic types are automatically converted to corresponding Java classes.

Rserve is very versatile, since it poses no limit on the facilities of R used. Although Rserve allows the execution of all R commands, the user should avoid any commands involving the GUI or console input, since Rserve has no console and there is no guarantee that it has any GUI at all. An exception are applications that provide their own copy of Rserve and have control over the way Rserve is started. Typical uses of Rserve include interactive applications using R for model construction (see KLIMT project) or web-servlets performing online computations.

As of now only basic types, such as numbers, strings and vectors hereof, can be assigned directly to R objects. The framework allows the transfer of arbitrarily complex types supported by the `REXP`, but the Rserve side is not fully implemented yet. Only transfer of evaluated objects supports all common expression types.

Rserve currently provides two client implementations: for Java and C languages. The Rserve protocol is well defined and allows the implementation of further clients in other programming or scripting languages when needed.

Rserve was tested on Linux, Mac OS X and Windows operating systems. The Windows version is the only restricted one, because there is no possible way of spawning new instances of R quickly. If all a Windows application needs is non-concurrent Rserve connections, it can provide its own copy of the Rserve binary, which will automatically find the last installed R and use it for computations, preventing clashes with other applications.

The Rserve project is released under GPL software license, which means that it can be modified or enhanced if necessary. The current Rserve is already used by several projects and is being enhanced for their needs.

References

[1] The omegahat project (`http://www.omegahat.org`).

[2] John M. Chambers. *Programming with Data. A Guide to the S Language.* Springer-Verlag, NY, 1998.

[3] Duncan Temple Lang. The omegahat environment: New possibilities for statistical computing. *JCGS*, 9(3), 2000.

[4] Tomoyuki Tarumi Masaya Iizuka, Yuichi Mori and Yutaka Tanaka. Statistical software vasmm for variable selection in multivariate methods. In *COMPSTAT 2002 Proceedings in Computational Statistics*, pages 563–568. Physica, Heidelberg, 2002.

[5] R Development Core Team. Writing r extensions (`http://cran.at.r-project.org/doc/manuals/r-exts.pdf`).

[6] Simon Urbanek and Antony R. Unwin. Making trees interactive - klimt. In *Proc. of the 33th Symposium of the Interface of Computing Science and Statistics*, 2001.