DSC 2003 Working Papers
(Draft Versions)
http://www.ci.tuwien.ac.at/Conferences/DSC-2003/

SciViews: an object-oriented abstraction layer to design GUIs on top of various calculation kernels

Philippe Grosjean

phgrosjean@sciviews.org

Abstract

SciViews aims to provide a unique, polymorph object-oriented interface (called "Plug") to different data analysis software like R, Octave, Scilab, Matlab, Splus, Ox, Mathematica,... It is thus possible to develop (Graphical) User Interfaces independently to the calculation kernel and adapt them easily to other systems. Moreover, the same GUI can possibly access several kernels simultaneously. A working prototype is developed in Visual Basic for Windows, but other implementations of this concept are possible. A plug to connect R is provided. Other plugs are in various stages of achievement. Several clients are also in development, including SciViews compatible clients for office programs (Excel and Word) and a versatile GUI with menus, toolbars, docking windows, a complete IDE and an integrated reporting system for generating rich-formatted texts with graphs called "Insider".

1 Introduction

Free data analysis software like R [1] (see http://www.R-project.org/), Octave [2] (see http://www.octave.org/) or Scilab (see http://www.rocq.inria.fr/scilab/) offer a command line interface (CLI). CLI has proven to be a flexible and efficient way for power users to interact with the calculation kernel. However, there is a steady demand [3] for a more intuitive graphical user interface (GUI) for some users, mainly occasional ones. Most commercial software like Splus, SPSS, SAS, Matlab, Mathematica, ... propose feature-rich GUIs.

There are several difficulties in building a GUI for a data analysis software originally designed to be used through a CLI. Some are technical, other ones are

conceptual problems. Major technical issues with CLI-designed software relate to the main loop (or event loop). With a CLI, command submission is synchronous: no interaction with the interface is allowed during calculation. Since a GUI is usually event-driven, and possibly interacts with the user or with other programs at any time (for instance, to redraw windows when they are resized or moved), it is not acceptable to freeze the UI completely during calculation.

For conceptual issues, the kind of GUI (spreadsheet, notebook, IDE-like, ...) has a major influence on the way the user interacts with the calculation kernel. The look-and-feel, and also extra features provided by the GUI can also range from extremely useful helping tools to annoying and useless gadgets. It seems that the way the GUI actually drives the user, and especially rather inexperienced statisticians can have a significant impact on the analyses. If the user interface is extremely easy to use, for instance, providing a dialog box with default values and an 'OK' button, there are chances that most inexperienced users run meaningless analysis. This is because such an interface does not require understanding of underlying calculations to produce "results". On the contrary, an UI requiring a stepwise constructive elaboration of the analysis, like S language keyed at the command line is by far less prone to such a side-effect. Moreover, the command history associated with the CLI allows easy recording and replaying of the session, a critical feature for repetitive tasks or for teaching (because it eases verification of student's works). This raises the question whether an intuitive and easy to use GUI is even desirable for data analysis software. Yet, GUI can simplify life of the user in many ways (context-sensitive help, explorers, completion lists, menus or shortcuts to frequently-used features). Thus, due to this ambiguity –beginners or occasional users want easier GUIs, but they learn better habits with a bare CLI–, the "perfect" GUI for data analysis software perhaps remains to be designed. It is probably a subtle balance between several contradictory requirements.

The main motivation for creating SciViews is to raise the opportunity to design easily different kind of GUIs on top of various calculation kernels, in order to encourage experimentation of concepts. The second major motivation relates to the other side: the calculation kernel. Several ones exist, with their particularities, their strengths and weaknesses. Therefore, not a single calculation kernel is able to answer all questions equally well. Having the same, familiar GUI accessing different kernels is a big advantage in this context. Unfortunately, such a polyvalent GUI does not exist. ESS (Emacs Speaks Statistics, [4]) introduced this attractive concept of one UI capable to drive various different statistical kernels. However, Emacs is a very particular UI, not really tailored to design GUIs (it is not its purpose). SciViews aims thus to fill the gap. The present paper exposes its concept and introduces a SciViews prototype (available at `http://www.sciviews.org/`).

2 Abstraction of the Kernel - GUI Interface

The key concept of SciViews is to provide a common, but polymorph object-oriented interface between the calculation kernel and the (graphical) user interface whatever the kernel and the GUI, in order to achieve their mutual exchangeability. This supposes an intermediary layer to translate interactions like inputs and outputs into a common set of methods and properties. In the SciViews jargon, we call this intermediary layer a "Plug" (see Fig. 1). Communication between the GUI and the plugs is centrally managed using an additional component: Xchanger.

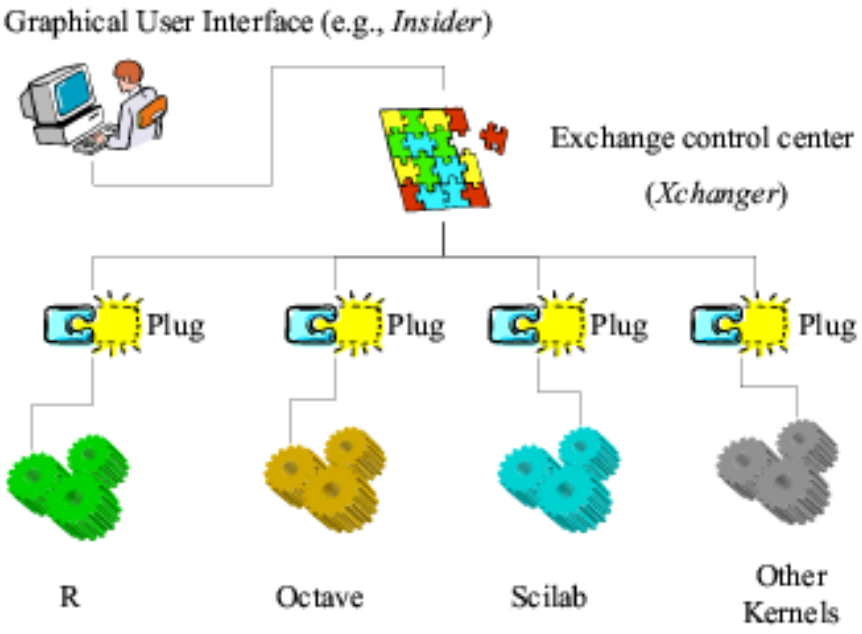

Figure 1: A GUI connects to various kernels using "Plugs".

Adding this intermediary layer between the UI and the calculation kernel is obviously resource consuming. Such an architecture is thus not pertinent when the data flow between the UI and the kernel(s) is large or when performances suffer no compromise. However, since it aims to substitute a typical command line interface, this gives the conditions of use: the equivalent of a command line string issued by the user for the input, and the equivalent of a few screens of textual results for the output, with potentially alternate means to exchange larger amount of data that will not flow through the plug (graphics, large matrices, ...). To define the limits of usability of such a system, we will accept solutions where:

$$\text{computation time} >> \text{transaction time}$$

or, in the case of very short computation time:

$$\text{number of transactions. sec}^{-1} < 10$$

with: 'computation time' being the time required to process the command inside the kernel and 'transaction time' being the processing time used by additional layers placed between the GUI and the kernel. The first condition applies for computer-intensive commands. It indicates that additional overhead due to the intermediary layer should remain proportionally negligible. The second condition is a limitation imposed for transactions that require very little computation in the kernel. A rapid succession of such transactions between the GUI and the kernel would saturate the processing unit with message passing through the intermediary layer. That second condition fixes thus the highest sustainable flow of transactions that the system can

guarantee. We arbitrarily place the limit at 10 transactions per second, that is, indeed much more commands than a user can issue per second!

A prototype is developed to check the concept and to verify that it satisfies these conditions. It is written using a RAD tool: Visual Basic 6 professional. A simple GUI client ("Test Xchanger") implements a few tests: (1) it can plug to a R kernel and initiate computer-intensive transactions, (2) it can spawn 10 nodes that each trigger transactions randomly in order to verify that concurrent use of the resources is possible, (3) it can trigger 10 transactions per seconds to check CPU usage in the second condition, and (4) it can fire as much transactions as possible per time unit (stress test) to push the system to the limits. "Stats Xchanger", another simple SciViews client, calculates the number of transactions per time unit actually achieved with a given test. Fig. 2 illustrates a test of the second condition performed on a Pentium IV 1.6 Ghz with 1 Gb memory (but less than 256 Kb is actually used) under Windows XP professional: the total CPU usage remains lower than 25%, leaving about 75% of the CPU resources free for computations. Of course, with a lower flow of transactions, that is, during usual conditions of utilization, the CPU usage by the additional SciViews layer is much lower: just a few percent, leaving much of the processing time for kernel's calculations. The prototype meets thus the two conditions.

3 Insider, a Versatile GUI Client

Several SciViews client GUIs are already planned: beside the simple test client described here above, addins for Excel and for Word are in development. We also initiate an ambitious client called "Insider". Insider provides a modern, highly flexible and customizable GUI with menus, toolbars, dockable windows, objects and files explorers, an IDE for scripts editing and an integrated reporting system combining rich-formatting text and images. Indeed, the list of the kind of document Insider can manage is expandable, since new ones can be easily added. Better yet, each document can freely interact with an unlimited amount of different calculation kernels thanks to the object-oriented architecture of SciViews. Fig. 3 is a screenshot of Insider showing a session with a single R kernel connected.

4 Nodes Hierarchy, System Integrity and XchgrMon

Even in a complex situation where possibly different GUIs connect to various kernels on the same machine or through the net, we want to leave the maximum control to the user. Each involved process is connected to SciViews Xchanger through a special object called "Node". Nodes are further organized into separate "Workspaces". A visual control of this organization is required. Direct interaction like interrupting transactions, forcing kernels disconnection at any time, or even killing frozen processes without endangering the rest of the edifice are possible direct actions we want to offer to the user.

Xchanger monitor (XchgrMon) displays graphically the hierarchy of workspaces, nodes, plugs and connected kernels (Fig. 4). It displays also a dynamic report of currently running transactions (in the 'Activity' tab, see Fig. 3). It allows direct interaction with those items via context sensitive menus.

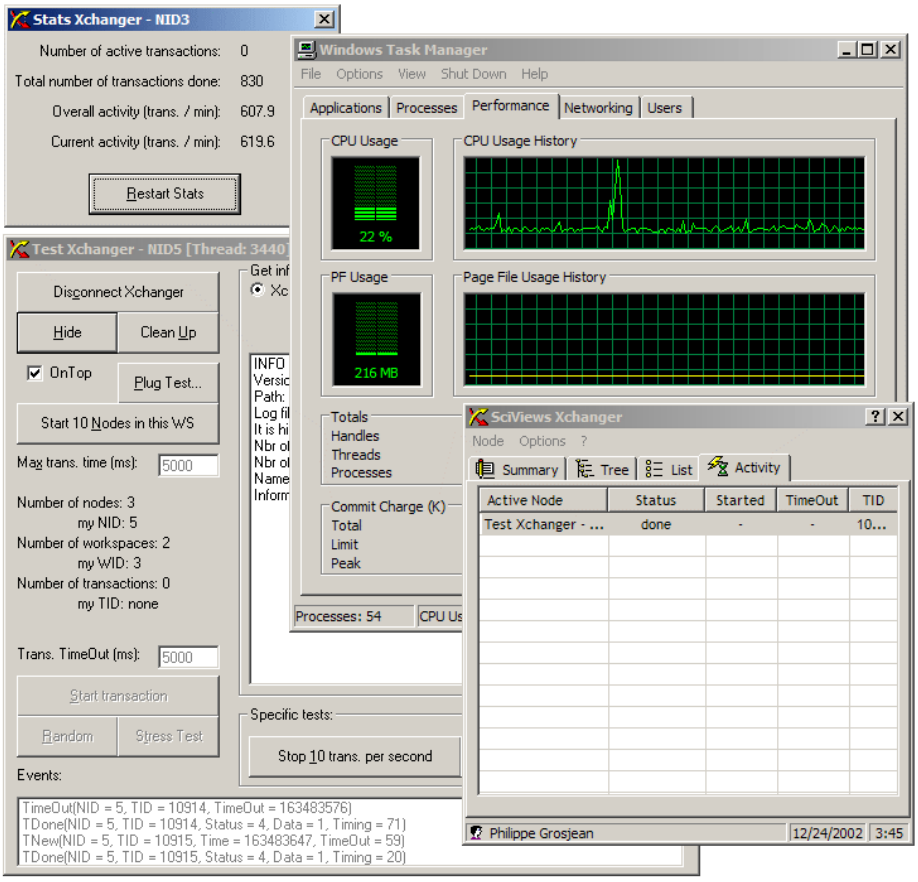

Figure 2: The test application (Test Xchanger) running 10 transactions per seconds. Stat Xchanger (top-left window) measures the actual number of transactions done per time unit. The 'Activity' tab of the 'SciViews Xchanger' monitor (see also section 5) is visible at the bottom-right. The Windows Task Manager is used here to record CPU usage, which remains consistently lower than 25%.

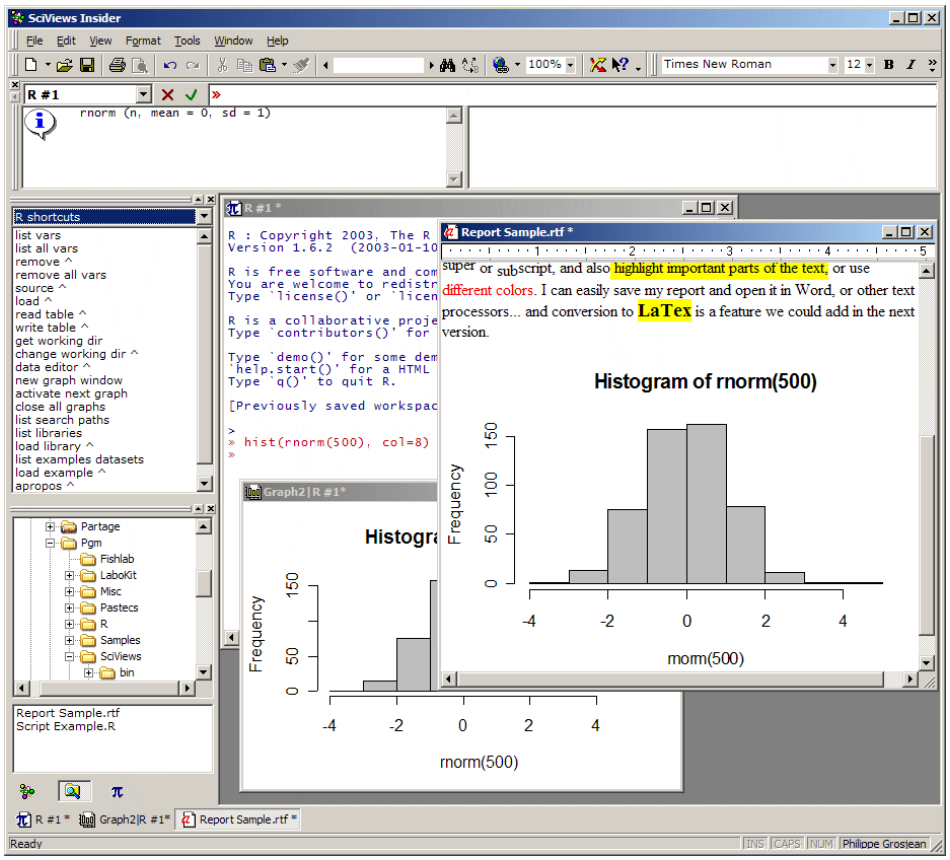

Figure 3: A sample Insider session with the integrated R console, a graph device and a reporting window. At the top, just beneath the menu and the toolbars, the command panel with a quick tip on the last used function (rnorm). A sample library of commands, and an explorer (here, the files explorer) are docked at left.

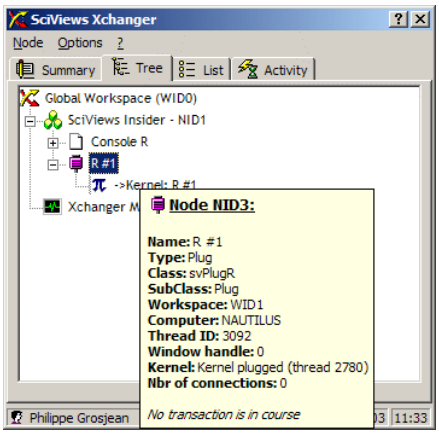

Figure 4: Xchanger monitor (XchgrMon) with the Workspaces, Nodes, Plugs and Kernels tree and an 'InfoTip' giving detailed information about the node NID3, that is, the node associated with the R Plug named 'R #1'(this tree corresponds to the Insider session depicted at Fig. 3).

Although these features are not implemented yet in the current prototype, Xchanger monitor will also controls the good working of Xchanger. It will verify it is "living" and processing requests from the nodes, and it will periodically check that the processes associated with the nodes are not frozen. It can kill any frozen process. Of course, clients must be programmed in such a way that they can recover from a killed connection at any time. This automatic clean up mechanism will ensure that dead items are removed as soon as possible from the SciViews node hierarchy, in order to maintain it fully operating.

5 Internal Architecture of SciViews

Until here, we mainly described externally visible features of SciViews. As a system aiming to design GUIs, these are major aspects. The internal organization of the different objects that have to collaborate together in order to achieve these results will be briefly outlined hereunder.

Fig. 5 shows how the SciViews architecture translates into object classes, in UML notations. The GUI (here it is Insider) contains one or more documents (Doc) that instantiate various plugs through a common, polymorph interface (IPlug). Each plug connects to a separate kernel. For instance, the class PlugR, a particular realization of IPlug connects to the R kernel. All object instances that participate in data exchange between the GUI and the kernels (except kernels themselves) instantiate also a node class, used as an entry point to Xchanger. The navigable (up and down) object hierarchy Node -> Nodes (collection of all node instances) -> Xchanger allows message passing between these nodes. Finally, XchgrMon allows the user to visualize and directly manipulate nodes, plugs and kernels.

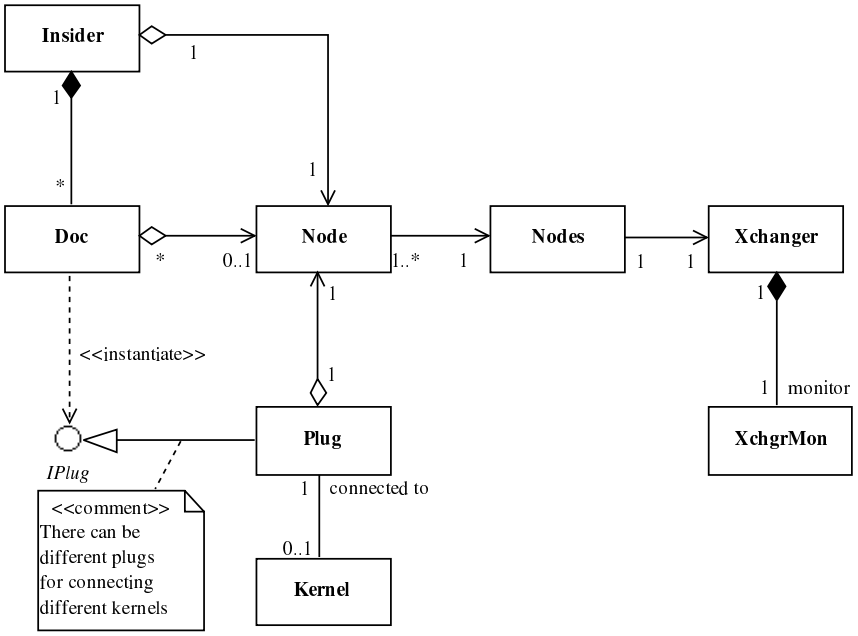

Figure 5: Class diagram for SciViews, when Insider is the client GUI.

5.1 The Plugs

Plugs offer the common interface by implementing the abstract IPlug class. Yet, each plug interprets differently commands like Connect(), Disconnect(), Evaluate(), ... depending on the kind of kernel they manage. An IPlug-compatible client can connect to various kinds of kernels, providing specific plugs exist. Plugs are programmed in order to require minimal changes in the code to adapt to a different kernel. The plug that is currently developed for R will be thus easily adaptable to other calculation kernels like Octave, Scilab, Ox, Splus, Matlab, Mathematica,...

Fig. 6 shows a sequence diagram during the lifetime of the plug, including connection and disconnection of the kernel. Interactions with Xchanger are also indicated. Xchanger collectively reports to all nodes changes like a node creation/destruction, or a kernel connection/disconnection. It associates also a NID (unique Node Identifier) to each node.

Execution of the kernel in another process necessitates careful controls and synchronizations. In particular, Connect() and Disconnect() require several verifications (does the kernel's executable exist? Is it the correct version? Is it starting when required? Is it closing gracefully after disconnection? Etc...). These additional steps are internally managed by the plug and remain transparent for the client.

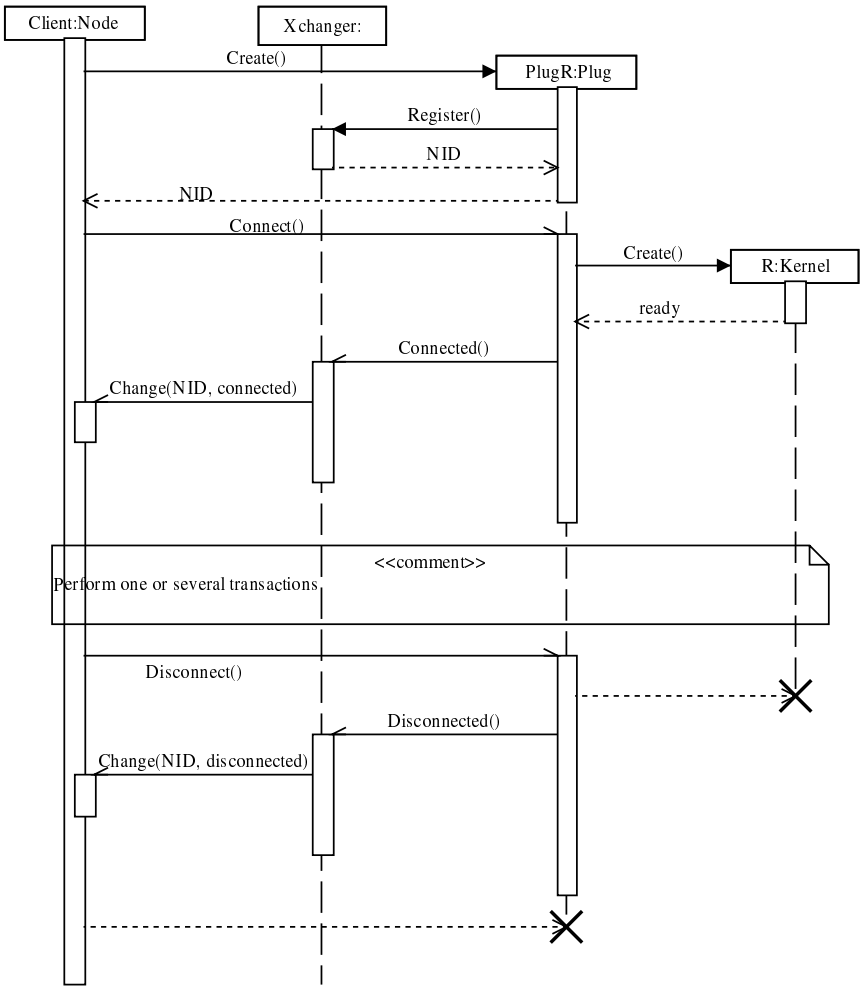

Figure 6: Sequence diagram of the plug lifetime with kernel connection and disconnection.

5.2 Asynchronous Management of Transactions

Just as Connect() and Disconnect(), Evaluate() is more complex than with a CLI. A CLI is usually synchronous: the user must wait for the completion of the calculation before further keying commands (Fig. 7). The console appears frozen during process of the command by the kernel (a very limited set of commands are still possible and handled in a special way, like Ctrl-C to interrupt the current calculation).

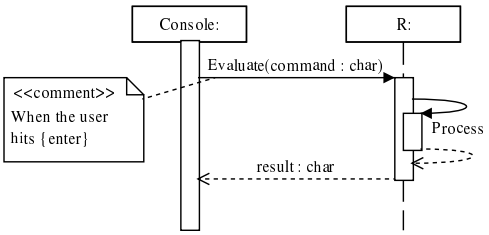

Figure 7: Sequence diagram for a typical "transaction" with a CLI.

With an event-driven GUI, it is not acceptable to freeze visible windows during (long) calculations. This is the event loop problem evoked in the introduction. The windows still have to react to system and user events. An asynchronous communication scheme is thus suitable here, but it is more complex that the sequence presented in Fig. 7. It requires to manage time outs in order to verify that the kernel processes the command and returns results in due time. This is partly the role of an additional class, "Transaction", to define and manage the context of this exchange.

Fig. 8 is the complete sequence diagram for a transaction between any client and the R kernel through PlugR. This is a sequence where the transaction is successful, but various cases of failure are, of course, also managed. It operates as follows. After successful connection to a kernel (see Fig. 6), a node sends an Evaluate(command) message to the plug. The plug informs Xchanger that a new transaction is initiated, and gets a unique identifier for that transaction (a TID). The plug then switches a flag to busy=true. In this state, it cannot process other commands until it returns to busy=false state, that is, when the current transaction is either fully processed, or when it is interrupted. The plug returns the TID to the node (or 0 if it is busy or not connected to the kernel). If the transaction is initiated, the plug sends the command to the kernel, usually with minor changes (for instance, after verification of a termination character at the end of the string), and the kernel starts processing it. The plug informs Xchanger that the transaction is now in 'processing' state, and Xchanger sends a similar message to the nodes.

Once the kernel has done its computation and has returned results to the plug, the latter warns Xchanger that the transaction is 'done', and Xchanger passes the message to all nodes. This message has an additional argument indicating which kind of output data is available (text, numerical value(s), image, file, error message,...). The node can then selectively ask results to the plug that returns them, possibly after translation into a format manageable by the client. Once this is done, the plug returns in a busy=false state and it is ready to process another transaction.

Xchanger provides a mechanism to automatically interrupt too long transac-

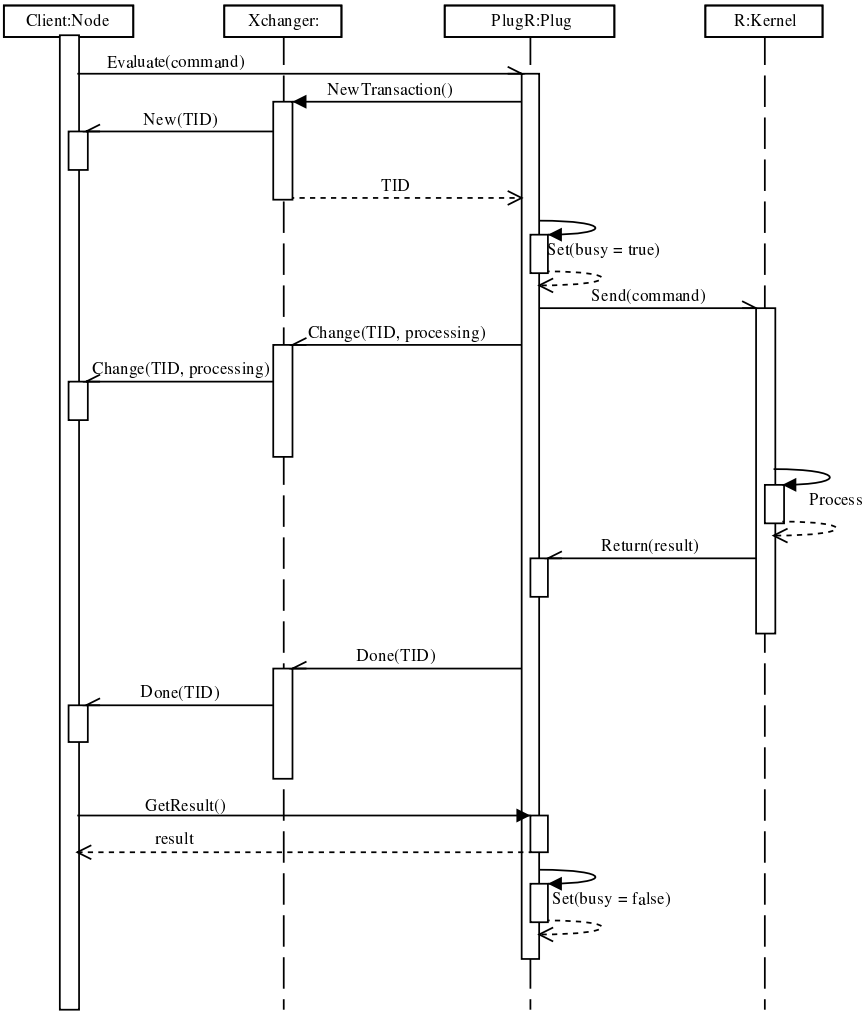

Figure 8: Sequence diagram of an asynchronous transaction in SciViews.

tions. When the client initiates a transaction, it provides two time out values: a processing time out, and a reception time out. The processing time out is the maximum allowed time to wait between the beginning of the transaction and the completion of the calculation by the kernel (that is, when the kernel starts sending back results). The receiving time out is the maximum delay permitted between the beginning and the end of output returned by the kernel. If one of these time outs occurs while the transaction is, respectively, in 'processing' or 'receiving' states, then Xchanger warns the plug and the client node and they can in turn take required action (cleanly stop the transaction or retry it, for instance).

The IPlug interface provides much more features, including full control of the child windows created in the kernel process (graphs, dialog boxes, ...) but these are not detailed here.

6 Conclusions and Perspectives

SciViews focuses on providing a common object-oriented communication between different GUIs and calculation kernels. A flexible GUI designed for data analysis (Insider) is also provided. Some features that ease access to statistical analyses thanks to object explorers, electronic reference cards (that we call "command libraries"), menus and dialog boxes and also interactive graphs can be developed within Insider. Although the whole software is just a prototype, it is a working "proof-of-concept". It is not platform-independent (Visual Basic and COM on which the prototype relies are proprietary developments of Microsoft, and they run only under Windows). A different implementation –in Java, or C++/CORBA– would be required to run on Unix/Linux or MacOS. Nevertheless, we will continue to develop the current prototype in the near future, because there is an urgent need for a R GUI under Windows.

Obviously, many aspects require optimization, including the way plugs communicate internally with the kernels (we used the ESS mode of R for the current PlugR). However, interlanguage communication libraries are rapidly evolving for data analysis software like R (see the Omega Hat project, `http://www.omegahat.org`). Since this communication process is encapsulated inside the plugs, such changes or optimizations could be done hopefully without touching at the rest of the SciViews architecture, and more particularly, without modifications of GUI clients.

The R Plug is complementary to the R COM server and client independently developed [5]. The R Plug is much more complex because it manages exchanges asynchronously, it has to cope with a complete object hierarchy (workspaces, nodes,...), and it is intended to be more robust against failures in the client or the server. The R COM server thus remains much simpler to use outside of the SciViews context. Due to these differences, we regard both the R Plug and the R COM server as two complementary products. Yet, it should be profitable to homogenize the syntax of their methods and properties as much as possible.

We hope that SciViews will raise opportunities and interests to experiment various kind of GUI metaphors (menu/dialog box, spreadsheet, notebook, object explorer, ...) in the context of data analysis, with R, but also with other free systems like Octave or Scilab. A more efficient, yet intuitive man-machine interfaces could perhaps emerge from such experimentations in the particular field of data analyses.

7 References

[1] Ihaka, R. and Gentleman, R., 1996. A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299-314.

[2] Eaton, J. W., 2001. Octave: past, present, and future. In Kurt Hornik and Friedrich Leisch, editors, *DSC 2001 Proceedings of the 2nd International Workshop on Distributed Statistical Computing*, http://www.ci.tuwien.ac.at/Conferences/DSC-2001/. ISSN: 1609-395X.

[3] Dalgaard, P., 2001. The R-Tcl/Tk interface. In Kurt Hornik and Friedrich Leisch, editors, *DSC 2001 Proceedings of the 2nd International Workshop on Distributed Statistical Computing*, http://www.ci.tuwien.ac.at/Conferences/DSC-2001/. ISSN: 1609-395X.

[4] Rossini, A., Mächler, M., Hornik, K., Heiberger, R. M., and Sparapani, R., 2001. Emacs Speaks Statistics: A universal interface fro statistical analysis. *Report 164, Department of Biostatistics, University of Washington*. URL http://software.biostat.washington.edu/statsoft/ess/ess~techrep.pdf.

[5] Baier, T., and Neuwirth, E., 2001 Embedding R in standard software, and the other way round. In Kurt Hornik and Friedrich Leisch, editors, *DSC 2001 Proceedings of the 2nd International Workshop on Distributed Statistical Computing*, http://www.ci.tuwien.ac.at/Conferences/DSC-2001/. ISSN: 1609-395X.