ILTAPÄIVÄSEMINAARI: Tilastollisten ohjelmistojen kehitysnäkymiä

Muste – the R implementation of Survo

Reijo Sund
National Institute for Health and Welfare,
P.O. BOX 30, 00271 Helsinki, Finland

ABSTRACT

Statisticians and other experts in data analysis need flexible tools for their work. R has become extremely popular software for such purposes. Survo is another flexible environment for statistical computing and related areas with many innovative properties and long history starting from the 1960s. The Muste project aims to implement the functionality of Survo as an R package. This article describes the background and current status of the project and discusses several technical details of advanced R programming requiring complex mixing of C, R, and Tcl/Tk-code.

Key Words: Survo, R programming, statistical software, Tcl/Tk, C, editorial approach, computational statistics

1. INTRODUCTION

Statistical analyses have an important role in quantitative empirical research. In addition, nowadays mathematical principles and theorems are not enough in statistics, but computational aspects have become more and more important. Computational statistics focuses on data processing and data analysis, and requires such special properties from the software that not any single statistical package can be comprehensive enough. Some may state that all they need is a general purpose programming language (such as C or Java), but in practice there are always some pieces of software that are used to achieve the intended results. This means that the software aimed to help statistical computing and related areas should provide a flexible working environment with excellent possibilities to freely extend the environment with new properties.

Importantly, many statistical software packages have been developed from the ”customer’s” point of view, where a customer is something else than an expert statistician with a sound background on mathematics and computing. From this point of view, it is likely that many statistical packages will become more and more ”user-friendly interfaces” to extremely complex analyses. In some cases that may be useful, and it certainly has become a profitable approach for business, but “easy” interface cannot replace the required statistical expertise. Moreover, limited possibilities to extend analyses beyond fixed choices, in tandem with high licensing costs combined with the slicing of important properties into several separate, expensive ”extension-bricks” restricts the usefulness of such software packages for a statistician. Fortunately, there are important outliers, such as R and Survo.

1.1 THE ORIGINS OF THE S LANGUAGE AND R

The history of S language started in 1976 at Bell Labs, and led to the development and implementation of statistical programming language that incorporated extensibility at a fundamental level (Chambers 2008, p. 475). After the implementation of S language in an open source project R (http://www.r-project.org) initiated by Ihaka & Gentleman (1996), R has grown and spread extensively and has become very popular among statisticians and other researchers. In spite of its popularity, the basic interface of R is quite crude. In addition, the complex manipulation of large datasets may pose problems and might be unnecessarily clumsy.

1.2 THE BRIEF HISTORY OF SURVO

The idea of statistical software package Survo (http://www.survo.fi/english) dates back to 1962 when a library of statistical programs was developed in the electronics department of the Finnish Cable Works (Mustonen 1967, pp. 1-3). Seppo Mustonen extended the idea during the years 1963-1964 and prepared a first proposal for a statistical programming system SURVO 64. Survo refers to the word ”survey” and additionally has a connotation with the Finnish word ”survoa” (=”to compress”) (Mustonen 1992a, p. 2). The first fully implemented system was SURVO 66 on Elliott 803 computer. The system allowed user to himself define the analyses to be performed (Alanko et al. 1968).

The SURVO 76 system was working with the Wang 2200 minicomputer (Mustonen 1977). The system provided an interactive environment and the use corresponded to conversation with the computer (Mustonen 1980a). The most remarkable single advance since that has been the invention of the *editorial approach*, which means that the interactivity is achieved by working with a text editor (Mustonen 1980b). Somewhat surprisingly the editor in Survo was originally developed to help the writing and printing of music sheets, that certainly was not the main purpose of an interactive statistical program. However, the editorial approach allowed such a flexibility and freedom to the user that it soon replaced the menu-driven interaction. In other words, the menu-based interface, that has become a standard in graphical operating systems since the 1990s, was found to be old-fashioned and restricting already at the beginning of the 1980s.

The editorial interface has been the heart of Survo systems since that. The PC-version SURVO 84C was already more like a statistical operating system (integrated environment) than just a program for statistical analyses (Mustonen 1992a). SURVO 98 was the extension to 32-bit version and SURVO MM the first Windows version of Survo. Since the introduction of an editorial approach in 1979, statistical computing and data analysis has been integrated as a part of writing and reporting the results of a research process. Similar ideas have been proposed more than twenty years later in the context of reproducible research (e.g. Gentleman & Temple Lang 2007).

In spite of its unique and early ideas, Survo is not well known, at least internationally. This is partially due to the facts that its development has been based on closed source policy, and the current version operates exclusively under Microsoft Windows.

2. MUSTE PROJECT

The Muste project was suggested in February 2009 by Reijo Sund. The main motivation for the project was that certain unique or otherwise useful properties of Survo system were not available in the form of an open source multiple platform software. As the R project had become very popular, offering an appealing environment for implementing open source statistical software for multiple platforms, it was a natural choice to test how it would be possible to produce Survo-look-alike functionality in this context. That has been an exciting adventure in R programming. There have been several challenges and it has sometimes been frustratingly hard and time-consuming to find solutions to the problems. The R extensions manual (R Development Core Team 2010) has been an invaluable source of information, but it could contain more examples.

In order to document the progress in the implementation of the Muste project and to provide simple examples in advanced R programming, we will describe the implementation process including some solutions to the technical problems encountered.

2.1 IMPLEMENTATION OF THE EDITOR

The first step in the implementation was to find a way to reproduce a new window for the editor as in the Survo system. It quickly turned out that the “tcltk”-package (Dalgaard 2001), that is a part of base R, would allow the control of additional windows in a flexible way with the Tcl/Tk. More precisely, the so called text widget made it possible to create a window containing text and control it appropriately. The text widget was easy to create and initialize using the commands in the “tcltk”-package. The actual problem was to remove the automatic functionality of the text widget and default key bindings that were not compatible with the Survo editor. The following extract of R code initializes the main window of Muste and disables certain default bindings:

```
require(tcltk)
# Create Window
.muste.ikkuna <<- tktoplevel()
# Disable automatic protocol for window deletion (alt-F4 or cross button)
tcl("wm", "protocol", .muste.ikkuna, "WM_DELETE_WINDOW",
    quote(cat("Delete protocol disabled!\n")))
# Disable resizing of the window
tkwm.resizable(.muste.ikkuna, FALSE, FALSE)
# Create font
.muste.font <<- tkfont.create(family="Courier",size=12) }
# Create text widget
.muste.txt <<- tktext(.muste.ikkuna,width=80,height=25,
                      foreground="black",background="snow",
                      wrap="none",font=.muste.font,undo=FALSE)
# Show the window containing the text widget
tkgrid(.muste.txt)
# Save the internal name of the widget
.muste.window<<-.Tk.ID(.muste.txt)
# Disable the default bindings of the text widget
bindings <- gsub("Text ","",tclvalue(tkbindtags(.muste.txt)))
tkbindtags(.muste.txt,bindings)
# Disable conflicting key bindings
tcl("bind","all","<Key-F10>","")
tcl("bind","all","<Alt-Key>","")
```

After the initialization of a text window, it was easy to write to the window and create bindings to keys so that a simplified look-alike version of the Survo editor could be implemented using R. It was also quite straightforward to include some basic commands to be used from the editor. In addition, it turned out to be possible to use the original non-interactive Survo-modules directly from the look-alike editor. That was possible because Survo has been developed so that many operations are actually separate executable (.exe) files and the real Survo editor also runs modules in a similar way with all required information between the “parent” and the “child” modules being transferred via temporary files.

Such tests were so promising that the next step was to test if it would be possible to use the C source code of Survo in the Muste project. Professor Seppo Mustonen (the developer of Survo) kindly provided all required source codes of Survo for such tests, starting with the Survo library functions (Mustonen 1989) and editorial arithmetics.

It was rather straightforward to compile the needed parts of the source code of Survo with a C compiler of the open source GNU Compiler Collection (GCC), and then call the produced stand-alone executable from the look-alike editor running in R in the same way as original Survo modules. That also demonstrated how it would be possible to compile modules for genuine Survo with an open source compiler instead of Microsoft's C compiler.

The next step towards a native R implementation was to utilize some source code of Survo directly in R as a part of an actual R package. That was quite straightforward and only minimal formatting changes had to be made to the original Survo C sources. The C code was called using .Call interface and Rinternals header file was used in the C side. Editorial arithmetics and a few other modules were tested and the calling of C-code from the package's dynamically linked library worked properly. Many mathematical functions in the Survo library were replaced with the corresponding R library functions that were described in the ”Writing R Extensions” manual.

As the look-alike editor programmed in R had very limited functionality, there was a need to replace it with the editor that was programmed in C as in genuine Survo. In practice this required that virtually all input/output (I/O) functions had to be reprogrammed, because they contained non portable Microsoft Windows specific code. Fortunately, Survo had been developed very smartly so that all platform specific functions were isolated, i.e. it was enough to replace these functions with corresponding ones for the current platform.

In this case, the actual platform was Tcl/Tk, but the interface for Tcl/Tk was via R as implemented in the “tcltk”-package. In other words, there was a need for executing Tcl/Tk code (with R interface) within C-code that was called from R, and that required communication between R-level objects and compiled C code. First task was to allow the evaluation of R-code within C-code. That was done by slightly modifying an example found in the “Writing R extensions” manual and by making a minimalistic wrapper for it:

```
#include <R.h>
#include <Rinternals.h>
#include <R_ext/Parse.h>
#include <stdio.h>

char komento[BUFFERSIZE];

SEXP Muste_EvalRExpr(char *cmd)
{
   ParseStatus status;
   SEXP cmdsexp, cmdexpr, ans = R_NilValue;
   int i;

   snprintf(komento,BUFFERSIZE,
   "if (inherits(try(.muste.ans<-%s,silent=FALSE), \"try-error\")) FALSE else TRUE",cmd);
   PROTECT(cmdsexp = allocVector(STRSXP, 1));
   SET_STRING_ELT(cmdsexp, 0, mkChar(komento));
   cmdexpr = PROTECT(R_ParseVector(cmdsexp, -1, &status, R_NilValue));
   if (status != PARSE_OK) {
       UNPROTECT(2);
       warning("Invalid call %s",cmd);
       return (R_NilValue);
   }
   for(i=0; i<length(cmdexpr); i++) ans = eval(VECTOR_ELT(cmdexpr,i),R_GlobalEnv);
   UNPROTECT(2);
   if (INTEGER(ans)[0]==FALSE) return (R_NilValue);
   ans = findVar(install(".muste.ans"),R_GlobalEnv);
   return ans;
}

int muste_evalr(char *cmd)
   {
   int retstat;
   SEXP status;
   retstat=1;
   status=Muste_EvalRExpr(cmd);
   if (status==R_NilValue) retstat=-1;
   return retstat;
   }
```

It is easiest to demonstrate the communication between R, Tcl/Tk and C with examples. The first example is a straightforward call to Tcl/Tk using R interface within C-code:

```
void sur_pos_window(char *wname,int x,int y)
    {
    snprintf(komento, BUFFERSIZE, "tcl(\"wm\",\"geometry\",%s,\"+%d+%d\")",wname,x,y);
    muste_evalr(komento);
    }
```

In this more complex example, a self-made R function

```
.muste.getscreendim <- function()
    {
    .muste.screen.width<<-as.integer(tkwinfo("screenwidth",.muste.ikkuna))
    .muste.screen.height<<-as.integer(tkwinfo("screenheight",.muste.ikkuna))
    }
```

is called within C-code, and the contents of R variables are translated into form that can be handled within C:

```
int sur_screen_dim(int *sizex,int *sizey)
    {
    SEXP avar=R_NilValue;

    snprintf(komento,BUFFERSIZE,".muste.getscreendim()");
    muste_evalr(komento);

    avar = findVar(install(".muste.screen.width"),R_GlobalEnv);
    *sizex=INTEGER(avar)[0];

    avar = findVar(install(".muste.screen.height"),R_GlobalEnv);
    *sizey=INTEGER(avar)[0];

    return(1);
    }
```

Similar communication techniques work fine also for more complex situations. Unfortunately, repeated calls to R-evaluation within C-code tend to be slow like the loops are slow in standard R programming. This slowness turned out to be very disturbing if there was a need to do a lot of small updates to the contents of a text widget.

While working in the Linux environment, it finally turned out that the direct call to “tcltk”-package’s dotTcl() function was enough to speed up the process to an acceptable level (i.e. the idea was to avoid unnecessary parsing in R). The problem was that although direct calls to unregistered functions in other packages were possible in Linux, that didn’t work in the Microsoft Windows environment because of different practices in dynamically linked libraries. This problem could have been solved easily, if the dotTcl() function had been registered as C-callable in the “tcltk”-package. As that was not the case, there was no other way than to use almost undocumented R_FindSymbol() function that is, however, included in the R-headers. There was also a need for the manual conversion of the parameters to suitable form for R, i.e. to LISP-style list. After some hours spent on exploring the sources of R it was finally possible to provide a working solution to the problem:

```
#include <R.h>
#include <Rinternals.h>
#include <R_ext/Rdynload.h>
#include <stdio.h>
#include <string.h>

int muste_window_existing=FALSE;
char muste_window[BUFFERSIZE] = "";
char tclkomento[BUFFERSIZE];
DL_FUNC RdotTcl = NULL;

int Muste_EvalTcl(char *komento, int ikkuna)
{
    SEXP alist,aptr;

    if (RdotTcl == NULL) RdotTcl = R_FindSymbol("dotTcl","tcltk",NULL);

/* Alternative for functions that are registered as C callable:     */
/*  if (RdotTcl == NULL) RdotTcl =R_GetCCallable("tcltk","dotTcl"); */
if (muste_window_existing==FALSE)
    {
    SEXP avar=R_NilValue;
    avar = findVar(install(".muste.window"),R_GlobalEnv);
    strncpy(muste_window,CHAR(STRING_ELT(avar,0)),BUFFERSIZE-2);
    strcat(muste_window," ");
    muste_window_existing=TRUE;
    }

    if(!ikkuna) strcpy(tclkomento,komento);
    else
    {
      strncpy(tclkomento,muste_window,BUFFERSIZE);
      strncat(tclkomento,komento,BUFFERSIZE-1-strlen(muste_window));
    }
    PROTECT(alist = allocList(2));
    aptr=alist;
    aptr=CDR(aptr);
    SETCAR(aptr, mkString(tclkomento));
    RdotTcl(alist);
    UNPROTECT(1);
    return(1);
}
```

This function finds the entry point to RdotTcl() function and constructs a suitable LISP-style list for parameters to be passed to the function. It also adds the name of the Tcl/Tk-window if needed. The most important use for this approach was the function for writing strings to the text widget:

```
int write_string(char *x, int len, unsigned char shadow, int row, int col)
    {
    snprintf(komento,BUFFERSIZE,"delete %d.%d %d.%d",row,col-1,row,col-1+len);
    Muste_EvalTcl(komento,TRUE);
    snprintf(komento,BUFFERSIZE,"insert %d.%d \"%s\" shadow%d",row,col-1,x,shadow);
    Muste_EvalTcl(komento,TRUE);
    return(len);
    }
```

This function first makes room for the string by deleting existing text from the widget and then inserts new text to the place of old. Here the name of the Tcl/Tk window is needed. Another example is a sleep function that waits for a given time in milliseconds so that the waiting thread does not waste all resources:

```
void muste_sleep(int time)
    {
    char buf[BUFFERSIZE];
    snprintf(buf,BUFFERSIZE,"after %d",time);
    Muste_EvalTcl("update idletasks",FALSE);
    Muste_EvalTcl("update",FALSE);
    R_FlushConsole();
    R_CheckUserInterrupt();
    Muste_EvalTcl(buf,FALSE);
    }
```

Sleep function was important for implementing an event loop inside the C-code that also detected the keyboard and mouse events of Tcl/Tk. There was also a need for an appropriate event handler so that all keyboard and mouse events recorded the event time and type that could be peeked using the C-code. The first version of the event loop used Survo source almost directly, but it had a drawback that it was not possible to use R directly from its terminal (or GUI), because R was busy for running the called C-code.

In order to free R while the editor was idle, there was a need for parallel event loops. In the current implementation, the event loops are not really separated, but simulated so that a short C-code checking the event status is repeatedly called with the help of “tcltk”-package’s possibilities. This solution may not be the most elegant one, but it works, which is currently the most important criterion for the implementation.

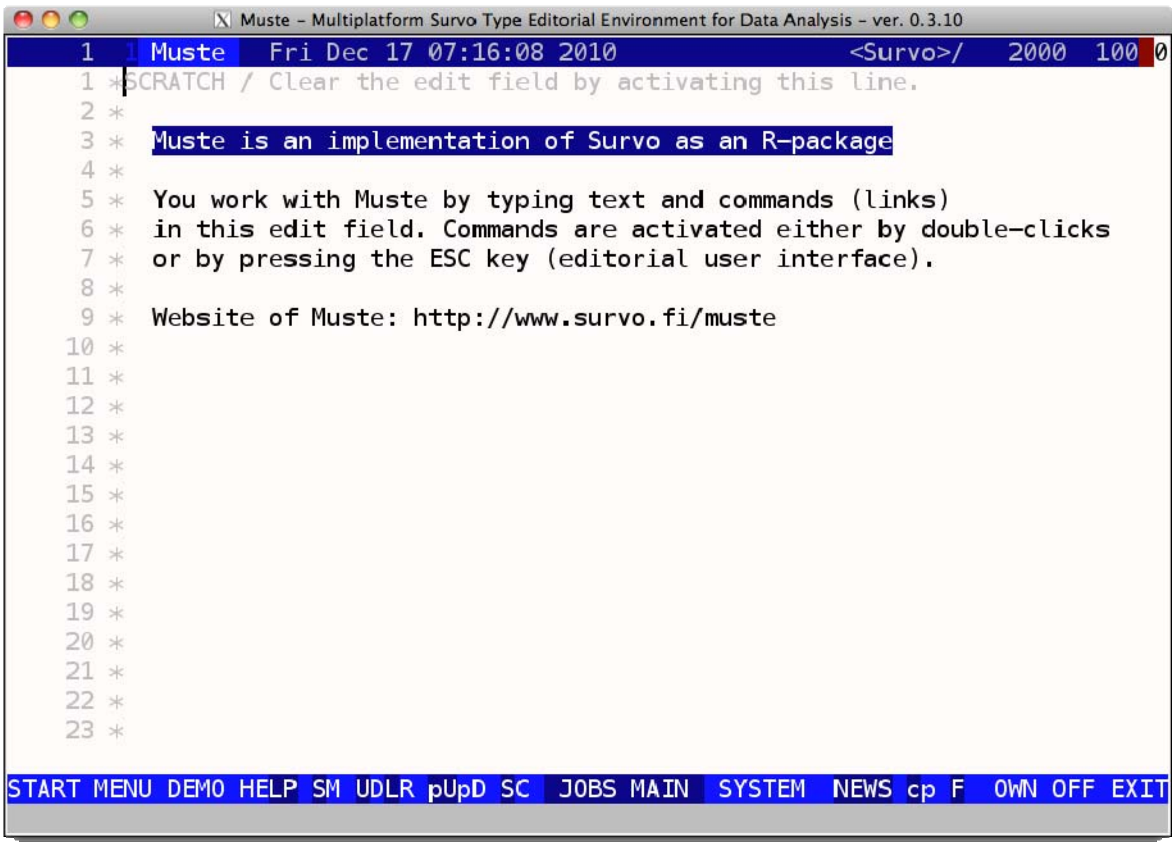
2.2 THE CURRENT STATUS OF MUSTE

At the moment of writing this, Muste (version 0.3.10) contains all basic functions of the Survo editor and editorial arithmetics. Also the support for sucros (Survo macros) is implemented. Most of the actual Survo modules are, however, not implemented yet. It is expected that the implementation of ”passive” modules such as the ones for matrix interpreter and PostScript printing will be very easy. The modules requiring a lot of file handling may be somewhat more challenging because of the non-portability of Windows style paths. It also seems that at least in some 64-bit Mac OS X systems 64-bit longs are causing some unexpected issues in structures and in some file functions. In addition, the basic functions and approaches to addressing the screen graphics will probably require some work, because it is not possible to copy that functionality directly from the Survo sources.

The development of Muste has taken place in different platforms. Initial versions were prepared with Debian-based Linux Ubuntu and Fedora-based Linpus Lite Linux. Some work has also been conducted under the Microsoft Windows Vista and Windows XP. Currently, we use mainly 64-bit Snow Leopard Mac OS X as a development platform. It has been a pleasure to notice how easy it is to prepare multiplatform R packages - the amount of issues requiring different solutions on different platforms has been negligible.

2.3 WORKING WITH MUSTE

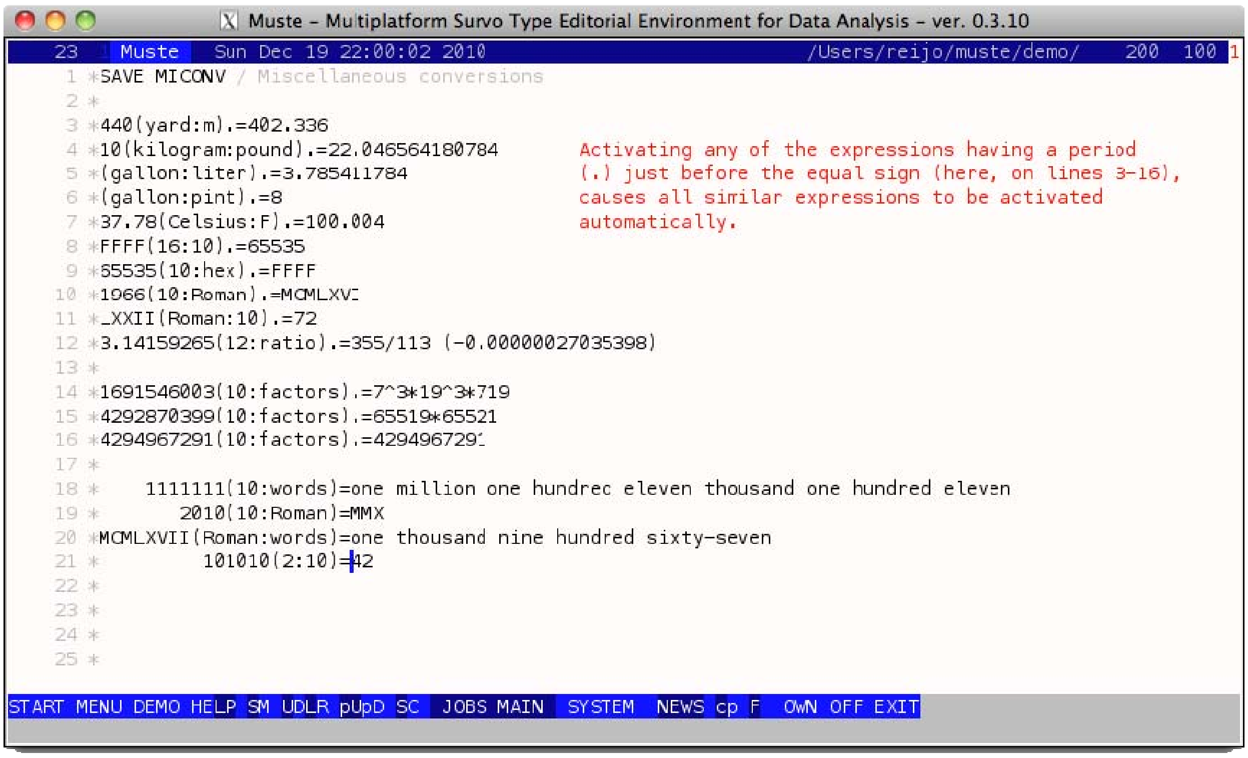
A screen capture of Muste window (in Mac OS X) after starting Muste with the command “library(muste)” in R is as follows:



```
Muste - Multiplatform Survo Type Editorial Environment for Data Analysis - ver. 0.3.10
    1   1 Muste   Fri Dec 17 07:16:08 2010               <Survo>/   2000  100 0
    1 *SCRATCH / Clear the edit field by activating this line.
    2 *
    3 *  Muste is an implementation of Survo as an R-package
    4 *
    5 *  You work with Muste by typing text and commands (links)
    6 *  in this edit field. Commands are activated either by double-clicks
    7 *  or by pressing the ESC key (editorial user interface).
    8 *
    9 *  Website of Muste: http://www.survo.fi/muste
   10 *
   11 *
   12 *
   13 *
   14 *
   15 *
   16 *
   17 *
   18 *
   19 *
   20 *
   21 *
   22 *
   23 *
START MENU DEMO HELP SM UDLR pUpD SC  JOBS MAIN  SYSTEM  NEWS cp F  OWN OFF EXIT
```

It shows a part of the edit field where a user can freely write any text and commands as in any text editor. At the beginning of each line there is a line number followed by a control column that contains only asterisks (stars) in the example but could include other symbols. Different colors are obtained by using ”shadows” for the text. Technically, shadows are another layer of text that is reflected in different colors in the actual edit field. The shadows may have both visual and structural purposes in various operations. The only command in the above window is SCRATCH (that erases all contents of the edit field on the same line or below that in the edit field). Commands are activated by pressing activate key that is escape (ESC). The header line shows the current date and time, the working directory, the size of the edit field, and certain status information. The bottom line contains so called soft buttons that can be activated with a mouse. More basic information can be found on the website of Survo (http://www.survo.fi/english).

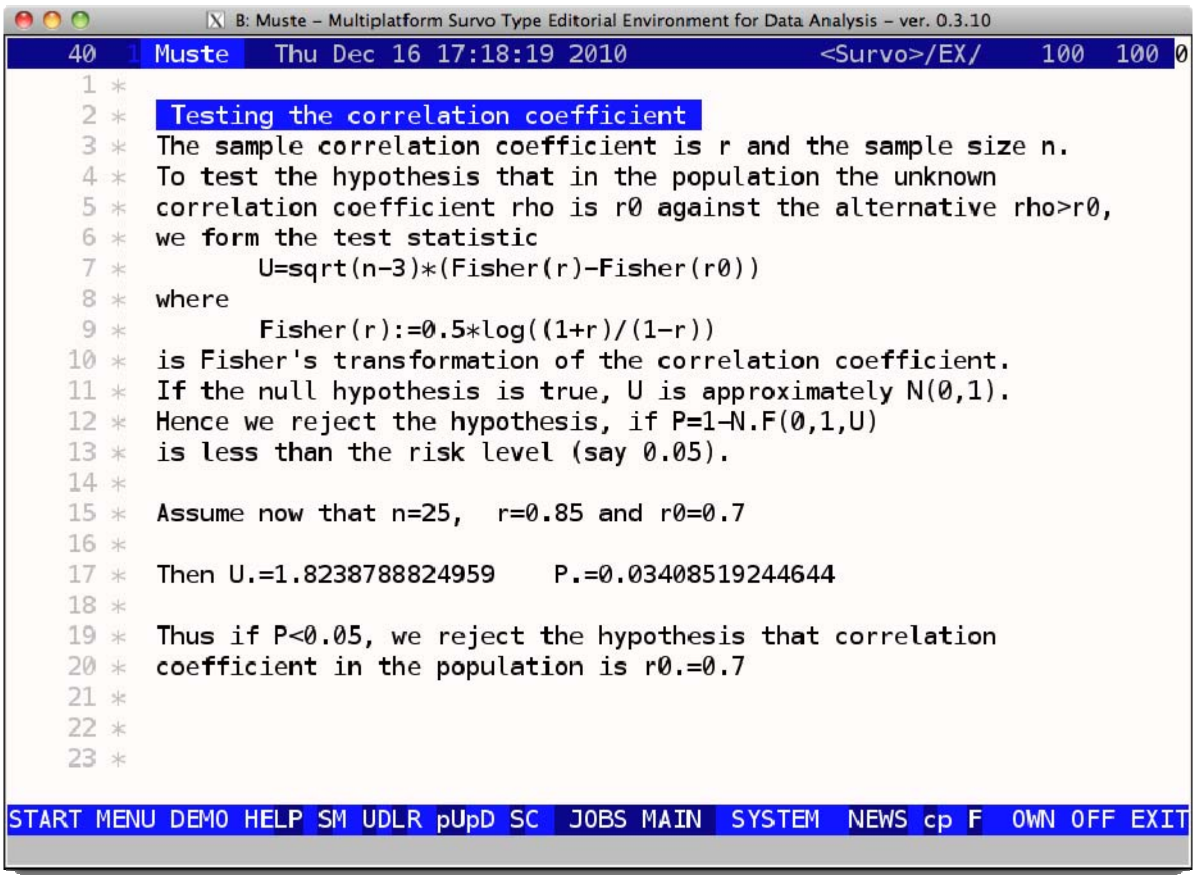
Some useful functionality of Muste can be seen in the following example, which demonstrates miscellaneous conversions (modified from a ready-made Survo tutorial):



```
Muste - Multiplatform Survo Type Editorial Environment for Data Analysis - ver. 0.3.10
   23  1 Muste   Sun Dec 19 22:00:02 2010                 /Users/reijo/muste/demo/    200  100 1
   1 *SAVE MICONV / Miscellaneous conversions
   2 *
   3 *440(yard:m).=402.336
   4 *10(kilogram:pound).=22.046564180784       Activating any of the expressions having a period
   5 *(gallon:liter).=3.785411784               (.) just before the equal sign (here, on lines 3-16),
   6 *(gallon:pint).=8                          causes all similar expressions to be activated
   7 *37.78(Celsius:F).=100.004                 automatically.
   8 *FFFF(16:10).=65535
   9 *65535(10:hex).=FFFF
  10 *1966(10:Roman).=MCMLXVI
  11 *_XXII(Roman:10).=72
  12 *3.14159265(12:ratio).=355/113 (-0.00000027035398)
  13 *
  14 *1691546003(10:factors).=7^3*19^3*719
  15 *4292870399(10:factors).=65519*65521
  16 *4294967291(10:factors).=4294967291
  17 *
  18 *    1111111(10:words)=one million one hundred eleven thousand one hundred eleven
  19 *       2010(10:Roman)=MMX
  20 *MCMLXVII(Roman:words)=one thousand nine hundred sixty-seven
  21 *         101010(2:10)=42
  22 *
  23 *
  24 *
  25 *

START MENU DEMO HELP SM UDLR pUpD SC  JOBS MAIN  SYSTEM  NEWS cp F  OWN OFF EXIT
```

Editorial arithmetic provides a powerful tool for making many types of calculations with an extremely flexible interface. The following example is about testing the correlation coefficient presented in detail elsewhere (Mustonen 1992b):



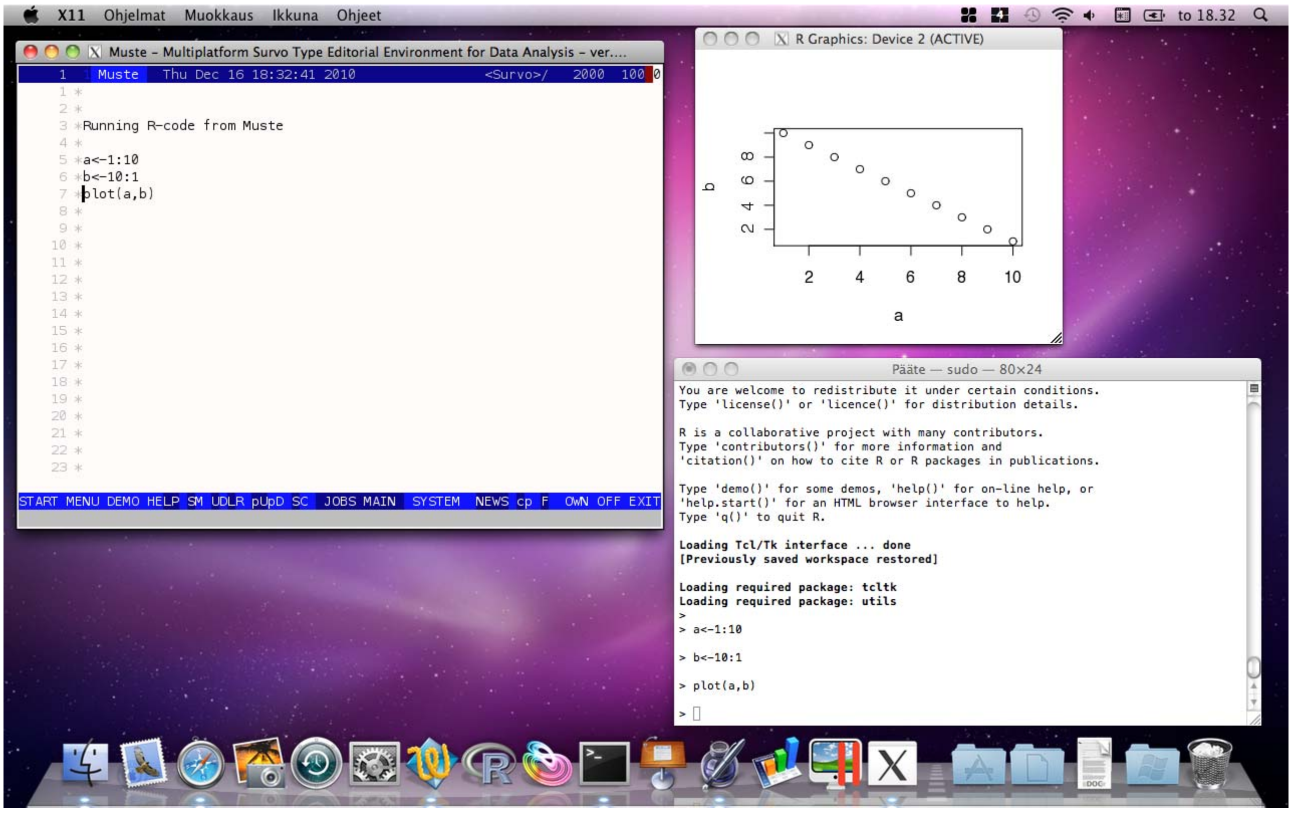
```
B: Muste - Multiplatform Survo Type Editorial Environment for Data Analysis - ver. 0.3.10
   40  1 Muste   Thu Dec 16 17:18:19 2010          <Survo>/EX/     100  100 0
 *
 *  Testing the correlation coefficient
 *  The sample correlation coefficient is r and the sample size n.
 *  To test the hypothesis that in the population the unknown
 *  correlation coefficient rho is r0 against the alternative rho>r0,
 *  we form the test statistic
 *         U=sqrt(n-3)*(Fisher(r)-Fisher(r0))
 *  where
 *         Fisher(r):=0.5*log((1+r)/(1-r))
 *  is Fisher's transformation of the correlation coefficient.
 *  If the null hypothesis is true, U is approximately N(0,1).
 *  Hence we reject the hypothesis, if P=1-N.F(0,1,U)
 *  is less than the risk level (say 0.05).
 *
 *  Assume now that n=25,  r=0.85 and r0=0.7
 *
 *  Then U.=1.8238788824959    P.=0.03408519244644
 *
 *  Thus if P<0.05, we reject the hypothesis that correlation
 *  coefficient in the population is r0.=0.7
 *
 *
 *

START MENU DEMO HELP SM UDLR pUpD SC  JOBS MAIN  SYSTEM  NEWS cp F  OWN OFF EXIT
```

As it is difficult to describe the benefits of the editorial approach in this kind of static text document, we suggest the reader to check some demos on the website Survo, for example http://www.survo.fi/demos/#ex6.

3. Co-operation between Muste and R

The purpose in the Muste project has been to implement the main parts of Survo functionality before switching to the development of other properties. Muste contains, however, some experimental properties that are not available in Survo. One example is the direct running of selected R-code from Muste with <control-R>. This means that Muste can also be used as a script editor for R. A simple example can be seen in the following screen capture, where some R-code has been activated directly from Muste:


X11 Ohjelmat Muokkaus Ikkuna Ohjeet
to 18.32
Muste – Multiplatform Survo Type Editorial Environment for Data Analysis – ver....
R Graphics: Device 2 (ACTIVE)
b
2 4 6 8
2 4 6 8 10
a
Pääte — sudo — 80×24
START MENU DEMO HELP SM UDLR pUpD SC JOBS MAIN SYSTEM NEWS cp F OWN OFF EXIT

```
1  Muste  Thu Dec 16 18:32:41 2010          <Survo>/   2000  100 0
*
*
*Running R-code from Muste
*
*a<-1:10
*b<-10:1
*plot(a,b)
```

```
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

Loading Tcl/Tk interface ... done
[Previously saved workspace restored]

Loading required package: tcltk
Loading required package: utils
>
> a<-1:10

> b<-10:1

> plot(a,b)

> 
```

Actually, it will be of primary importance to increase the co-operation between Muste and R in the future. Muste will provide an interface for R and include excellent data manipulation possibilities that can be used with data sets that may be challenging to be handled directly in R. For example, Muste does not keep all the data in the memory but handles them as files. As Muste is a native R package, it will be available for all users of R who may need or want to utilize the functionality it offers.

Of course there might be some unexpected challenges during the implementation, but so far it seems that it is feasible to carry on with the Muste project. The main goal is to release Muste as an open source R package. As far as we know, Muste will be one of the most many-folded R packages offering several extra properties and extensions to R as it will contain all essential functionality of the legendary Survo system.

Please visit the Muste website for up-to-date information: http://www.survo.fi/muste

ACKNOWLEDGEMENTS

The author is extremely grateful to professor Seppo Mustonen who has kindly provided all necessary source codes and tacit knowledge concerning Survo. The Muste project would not have been possible without his enthusiastic support and important contributions. The author would also like to thank Kimmo Vehkalahti for all his efforts to the Muste project.

References

Alanko T, Mustonen S, Tienari M. A Statistical Programming Language SURVO 66. BIT 8:69-85, 1968. DOI: 10.1007/BF01939330

Chambers JM. Software for data analysis. Programming with R. Springer Series in Statistics and Computing. Springer, New York, 2008. DOI: 10.1007/978-0-387-75936-4

Dalgaard P. The R-Tcl/Tk interface. Proceedings of the 2nd International Workshop on Distributed Statistical Computing (DSC 2001), March 15-17, Vienna, Austria, 2001.
http://www.ci.tuwien.ac.at/Conferences/DSC-2001/Proceedings/Dalgaard.pdf

Ihaka R, Gentleman R. R: A language for data analysis and graphics. Journal of Computational and Graphical Statistics 5:299-314, 1996. DOI: 10.2307/1390807

Gentleman R, Temple Lang D. Statistical Analyses and Reproducible Research. Journal of Computational and Graphical Statistics 16(1):1-23, 2007. DOI: 10.1198/106186007X178663

Mustonen S. Tilastollinen tietojenkäsittelyjärjestelmä SURVO 66. Monistesarja, Tampereen yliopiston tietokonekeskus, Moniste n:o 2, Tampere, 1967. [Statistical Data Processing System SURVO 66, Report no 2 of the Computing Centre in the University of Tampere].
http://www.survo.fi/publications/SURVO_66_Mustonen_1967.pdf

Mustonen S. SURVO 76, a statistical data processing system. Research Report No. 6. Department of Statistics, University of Helsinki, 1977.
http://www.survo.fi/publications/Research_Report_6_Mustonen_1977.pdf

Mustonen S. Interactive analysis in SURVO 76. Proceedings of the 4th Symposium on Computational Statistics, COMPSTAT, Edinburgh, Scotland, 1980. M.M. Barritt and D. Wishart, Editors, pp. 253-259. Physica-Verlag, Wien, 1980a.
http://www.survo.fi/publications/COMPSTAT_1980.pdf

Mustonen S. SURVO 76 EDITOR, a new tool for interactive statistical computing, text and data management. Research Report No. 19. Department of Statistics, University of Helsinki, 1980b.
http://www.survo.fi/publications/Research_Report_19_Mustonen_1980.pdf

Mustonen S. Programming SURVO 84 in C. SURVO 84C Contributions 3. University of Helsinki, Department of Statistics, 1989.
http://www.helsinki.fi/survo/c/

Mustonen S. Survo - An Integrated Environment for Statistical Computing and Related Areas. Survo Systems, Helsinki, 1992a.
http://www.survo.fi/books/1992/Survo_Book_1992_with_comments.pdf

Mustonen S. Editorial interface in Statistical Computing and Related Areas. Proceedings of the 10th Symposium on Computational Statistics, COMPSTAT, Neuchâtel, Switzerland, August 1992. Yadolah Dodge and Joe Whittaker, Editors, Volume 2, pp. 17-32. Physica-Verlag, Heidelberg, 1992b.
http://www.survo.fi/publications/COMPSTAT_1992.pdf

R Development Core Team. Writing R Extensions. Accessed Dec 20th, 2010.
http://cran.r-project.org/doc/manuals/R-exts.pdf