A Tcl-based SRS v. 4 Interface

Gijs Schaftenaar, Koen Cuelenaere, Jan H. Noordik and Thure Etzold

Abstract

A new SRS (Sequence Retrieval System) user interface has been developed for SRS v. 4. Key features are the support of simple character-oriented (ASCII, VT100) terminals by coding in Tel augmented by some dedicated Curses calls, support of graphics terminals in an X-Windows version by using the Tk extension to Tel, and support of a client/server environment by using the TDP extension to Tel.

The Sequence Retrieval System (SRS) is a powerful tool for the fast extraction of information from flat file libraries (Etzold and Argos, 1993) and has rapidly established itself as a major research instrument for the bio-informatics community. Internally the system employs a query language, which is user accessible through either a command-line user interface, 'getz', or a more user friendly, character-oriented window interface. For SRS versions up to release v. 3, this window interface supported VT100-compatible terminals. Because of major changes in the underlying SRS libraries, the v. 3 interface became fully incompatible with the most recent version of SRS (v. 4.x). Thus the many users with only a simple terminal/terminal emulator connection were either deprived of access to SRS, or were forced to use the ASCII WWW client LYNX. This prompted us to develop a character-oriented SRS v. 4 window interface with the look and feel of its SRS v. 3.1 predecessor and coded to be as library independent as possible to maintain compatibility with future SRS releases. In addition, some 'extensions' were coded to widen the applicability to graphics terminals and to a client/server environment.

At the time of preparation of this paper, the SRS interface described had been implemented in one form or another on most EMBnet nodes and on all the platforms given in Table II. The code has been stored at the EMBL in Heidelberg, where it will be available, with installation instructions and scripts, as part of the SRS distribution.

Interface specification and development tools

For the interface(s) code to be developed, the following specifications were defined. The code should:

- only use widely available and well-maintained public domain tools and libraries;
- eliminate the need for continued maintenance of older SRS releases;
- serve the large user-group which still has only access to simple ASCII terminals/terminal emulators;
- eliminate the need for renewed training in the use of SRS;
- in an extended form, be usable on X-Windows graphics terminals;
- eliminate the need for continuous interface modification, with successive SRS releases.

To meet these requirements, the interface presented in this paper has been coded in Tel, the Tool Command Language (Ousterhout, 1994). The character-oriented (ASCII) version employs some C-Curses calls for screen management. For the graphics version we have also used the Tk (X11 Toolkit) extension to Tel, and for the client/server version the TDP (Tel-Distributed Programming) extension. Tel is an interpreter language which can easily be extended with new commands. Of the publicly available extensions, Tk adds commands for graphical attributes like windows, buttons and scrollbars. The TDP extension (Smith et al., 1994) adds commands which enable communication between different computers in a network, using remote procedure calls (RPCs). The interface was coded using these tools and a custom-built C-Curses widget library. This library supplies the pseudo-graphics functionality using the character set of non-graphics terminals and is implemented on top of Tel. Although this dedicated C-Curses widget library still contains many hundreds of lines of code, it could be coded much faster than a completely general library. The necessary Curses extensions to Tel were taken from the literature (Kamp, 1994) and ported, as summarized in Table I, to the platforms summarized in Table II.

The ASCII interface srscurs constructed in this way has the look and feel of the previous SRS v. 3 interface and simultaneously all the 'hooks' as detailed in the
Table I. Curses extensions to Tcl

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>curses initscr</code></td>
<td>Initialize Curses package</td>
</tr>
<tr>
<td><code>curses endwin</code></td>
<td>Terminate Curses package</td>
</tr>
<tr>
<td><code>curses initscr</code></td>
<td>Initialize Curses package</td>
</tr>
<tr>
<td><code>curses endwin</code></td>
<td>Terminate Curses package</td>
</tr>
<tr>
<td><code>curses mode</code></td>
<td>Set terminal mode</td>
</tr>
<tr>
<td><code>curses newwin</code></td>
<td>Create new Curses window with</td>
</tr>
<tr>
<td><code>curses refresh</code></td>
<td>Refresh window</td>
</tr>
<tr>
<td><code>curses clear</code></td>
<td>Clear window</td>
</tr>
<tr>
<td><code>curses getch</code></td>
<td>Get character input from window</td>
</tr>
<tr>
<td><code>curses addstr</code></td>
<td>Write string to window</td>
</tr>
<tr>
<td><code>curses delete</code></td>
<td>Delete window</td>
</tr>
<tr>
<td><code>curses touch</code></td>
<td>Touch window</td>
</tr>
<tr>
<td><code>curses box</code></td>
<td>Put box around window</td>
</tr>
</tbody>
</table>

Table II. Supported platforms

<table>
<thead>
<tr>
<th>Platform</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC Vax/VMS and OpenVMS</td>
<td>only 'getz' and 'internal' mode; no solid client/server Tdp</td>
</tr>
<tr>
<td>DEC Alpha OSF/1</td>
<td>no solid client/server Tdp</td>
</tr>
<tr>
<td>DEC Ultrix</td>
<td>extensions available</td>
</tr>
<tr>
<td>Silicon Graphics IRIX</td>
<td></td>
</tr>
<tr>
<td>SunOS</td>
<td></td>
</tr>
<tr>
<td>Sun Solaris</td>
<td></td>
</tr>
</tbody>
</table>

Supported operating system versions are those supported by Tcl/Tk specifications. Figure 1 shows an exemplary query mask of this interface, for a sequence query, and in Appendix I a summary of the C-Curses window interface elements is presented.

With `srscurs` as a starting point but using the Tk extension to Tcl for screen management instead of C-Curses calls, an X-windows graphics version of the interface, `srstk`, was written. This version uses the extra functionality as supplied by the possibilities of a graphics screen. Figure 2 shows the same query mask as shown in Figure 1, but now with the database selection option which can be activated by mouse clicks.

While we were completing the coding of `srscurs` and `srstk`, a C-Curses version of Tk called CTk was released on Internet (Andrews, 1995). Although CTk is not yet a comprehensive library, we nevertheless incorporated this library in our interface, replacing our dedicated widget set, for another version of `srscurs`. But due to the incompleteness of CTk this interface, `srsctk`, lacks some of the functionality of our basic ASCII version. Figure 3 shows the query mask of Figure 2 in the `srsctk` view. As it is still...
Fig. 2. Query mask window of the srsctk interface.

character oriented, in this window the Tab key must be used to select a database.

Both srsctk/srsctk and srsctk can be executed in different run modes. These modes differ mainly in the way in which the SRS query engine is addressed. Details of these run modes are discussed in the following paragraph.

Implementation

Both the character-oriented interface versions srsctk and srsctk and the Tk X-Windows version srsctk build the same query expression to query a local SRS implementation. The query handling is either *external* through the ‘getz’ executable of SRS, or *internal* through SRS extensions to Tcl, in which case the SRS query engine is embedded in the Tcl language. For use with a remote SRS server, a client/server application, two run-modes are available: either through the DP extension to Tcl; or through the HASSLE protocol (Doelz, 1994)—executable ‘hgetz’.

For the *internal* run mode, Tcl had to be extended with a concise set of SRS commands, because standard Tcl does not support SRS calls. This extension is comparable to the Tk extension for graphics. In these SRS extensions to Tcl, as presented in Table III, we have included all important SRS functionalities. All added command names are preceded by *srs_* to indicate clearly their function.

Both the *external* (getz query engine) and the *internal* (Tcl embedded query engine) run mode of the interface(s) have their advantages and disadvantages. A disadvantage of the external mode is that the interface itself has to keep track of many of the performed actions. For example, query expressions have to be stored because ‘getz’ is a stateless query engine, i.e. it has no memory of previous queries. On the other hand the interface invokes every ‘getz’ query as a background process of which the output is redirected to a scratch file, which has the advantage that the interface is not vulnerable to query engine core dumps. Long-running queries can be aborted by sending a kill signal, e.g. a Control-C. We have implemented the ‘getz’ mode as the default run mode for all Unix platforms.

In the *internal* mode, abortion of queries is not possible

Fig. 3. Query mask window of the srsctk interface.
because this functionality is not supported by the SRS C library. Moreover the interface in this mode is vulnerable to core dumps generated by internal SRS library calls. Theoretically, the internal mode is faster than the 'getz' mode because there is no overhead of reading of scratch files. In practice the difference in performance is negligible. The internal mode is the default mode in our VMS implementation, since the initiation of a background process (or spawning) on VMS, as required in the 'getz' mode, is prohibitively slow.

Both the external and the internal run modes have a client/server equivalent. The equivalent of the internal mode employs the Tcl TDP network extension, resulting in the look and feel of the local srstk. The server runs a Tcl shell, augmented with the SRS commands of Table III. SRS queries generated by the client-end of the interface are communicated to the server via the TDP mechanism, which in turn executes them and communicates the result back to the client via the same mechanism. The implementation of this mode consists of three short Tcl scripts. The first one sets up the main server, which basically only listens and waits for a client to make a connection. If a connection is established the first script spawns the second one, a child server, which executes the SRS calls and communicates directly with the client. The third script is used by the client to set up the primary connection with the main server and subsequently the child server. This set-up is illustrated in Figure 4.

The client/server equivalent of the external run mode uses 'HASSLE getz' as an external query engine. Essentially in this mode, the SRS executable 'getz' is executed over the network, whereby the interface communicates with the query engine via reading/writing from/to a Unix pipe. Implicit in this mode is the impossibility to abort queries, but the protocol saves the trouble of having to set up a new server connection for each query.

Acknowledgements

The development of the interface described in this publication was supported by EMBnet funds under EU grant BIO2-CT944002. The authors thank Jack Leunissen for support in initiating this project.

Fig. 4. SRS interface client/server implementation.
A Tcl-based SRS v. 4 Interface

References

Some of the references below are only available as ftp-sites and/or e-mail addresses; a situation which is more and more frequently encountered if software is developed with the abundance of public domain tools retrieved from Internet.

Andrews,M. (1995) andrew@ccfadm.eeg.ccf.org Cleveland Clinical Foundation, Cleveland, Ohio, US.


Kamp, Poul-Hennig. (1994) Curses extensions to Tc[+], phk@data.fls.dk.


Smith,B. et al. (1994) Tcl-DP; distributed programming extension to Tcl/ Tk. Documentation file released with the TDP extension.

Received on September 4, 1995; accepted on December 8, 1995

Appendix

Syntax of the srscurs widget command to specify and display a query form

```
gform xpos ypos width height message fields_definition
```

where `xpos` and `ypos` are the screen coordinates of the upper left corner of the form; `width`, `height` define the size of the form in character units; `message` is one line of text to be displayed on the status line; and `fields_definition` is a Tcl list where the elements consist of a Tcl list describing the various fields.

A list describing a field has the following elements in sequential order:

1. `accelerator`—a one-letter code used to activate a field without use of the cursor.
2. `field_name`—a short string to characterize the name of the field which will be displayed on-screen.
3. `variable`—the name of the global Tcl variable to contain the value of the field.
4. `x,y`—coordinates of field relative to the form some special symbols are allowed: `++` or `--` of previous field plus one `=` or `--` of previous field.
5. `field_type`—can be any of the following: integer,real,lor2,YorN, string,upper,return,execute.
6. `field_length`—length in characters of the field.
7. `command`—optional Tcl command association with field-type 'execute'.

Example of a field list

```
set flist { \
  {Q {query (set) name} setname [23 ++] upper 20 {}} \ 
  {{(query expression:)} dum [50 =] \ 
  {X {xpress [6 ++] upper 65 {}}} \ 
  {{] {] dum [2 ++]} \ 
}
```

Syntax of the srscurs widget command to specify and display a menu

```
menu menu_name xpos ypos item_list
```

where `menu_name` is the menu identifier; `xpos`, `ypos` are the screen coordinates of the upper left corner of the menu; and `item_list` is a Tcl list where the elements consist of a Tcl list describing the various items.

A list describing an item has the following elements in sequential order

1. `accelerator`—a one-letter code used to activate a field without use of the cursor.
2. `label`—the visible label of the item.
3. `info`—the status line info describing the item in more detail.
4. `item_type`—can be any of the following:
   * command activation of this item will trigger execution of a command specified by the item_value
   * check toggle between the values 0 and 1 of the global Tcl variable specified by the item_value
   * radio set the value of the global Tcl variable specified by the item_value to 'label'
5. `item_value`—depending on the item_type, either a command to be executed or the name of a global variable.

Example of a menu item list

```
set alist { \
  {D {output directory} directory [24 ++] string 42 {}} \ 
  {N {file name} Fname [17 ++] string 49 {}} \ 
  {F {sequence format} Form [23 ++] execute 1 { \
    Menu Format 29 4 \ 
    [G GCG {GCG sequence format (1 sequence per file)} radio Format] \ 
    [P PIR {PIR sequence format (all sequence into one file)} radio Format] \ 
    [F Fasta {Fasta sequence format (all sequence into one file)} radio Format]; \ 
  query touch} \ 
); \
```