Tcl in a High-Throughput Biological Lab

Scott P. Hunicke-Smith & Dan Mosedale – Stanford Yeast Genome Project

Abstract

The implementation and use of a Tcl-based language for the control of a laboratory robot is described. Also discussed are ideas for the future use of Tcl and Tk in various laboratory roles. The enthusiastic acceptance of the current implementation, the portability of Tcl scripts and Tcl itself, and the network of support available lead us to believe that Tcl should enjoy more use in any high-throughput (i.e. automated) biochemical laboratory.

1 Introduction

The main goals of the Stanford yeast genome group are to sequence the genome of *Saccharomyces cerevisiae* (a popular strain of yeast) and to develop automated approaches which will ultimately prove useful in sequencing the human genome. At present, determining this genetic sequence is largely a manual, labor-intensive process involving large numbers of samples and biochemical process steps. Automation of well-established sample manipulation protocols is necessary in order to reduce costs and improve accuracy. In addition, ever-increasing numbers of samples require computerized tracking. Tcl/Tk may serve as a good foundation for future improvements in both these areas. To give a rough idea of the scope of this project, consider the statistics in Figure 1 (taken from current production at Stanford).

At 7500 “handling steps” per week, the yeast genome could be sequenced in 48 years, the human genome in 11,500 years. This illustrates the necessity of collaboration in sequencing, that sequencing technology must improve, and that an enormous amount of data must be handled.

Our goal is to double the production rate each year over the next 5 years, and Tcl/Tk may be a critical part of that increase. Our first use of Tcl has shown great promise; we have replaced commercial software with a Tcl command-line interpreter named SYGI (Stanford Yeast Genome Interface) which is used to drive a laboratory robot.

Other possible uses of Tcl include:

- New instruments with Tcl as the primary interface.
- Control of multiple Tcl-driven instruments via a Tk interface at the workstation level.
- Completion of data and sample flow control, by incorporating a commercial database into the production sequencing.

2 Current Use of Tcl

2.1 The Biomek in its Original Form

The Biomek 1000 laboratory robot and it’s accompanying side-loading assistant, the Biomek SL, were primarily intended to replace repetitive manual pipetting. With this simple goal in mind, Beckman Instruments created software which they thought would be usable by the biological community using the tools available at the time (about 1988). The net result was a program called Genesis which required very little computer knowledge to operate, though one still had to learn the menu-driven control structure of the program. Further, it simplified the programming environment by pre-defining many of the tasks, tools, and disposables which could be used with the instrument. This left the user with relatively simple but rigid control.

Apparently this interface worked quite well for most labs doing simple tasks, but as laboratory automation increased, users such as ourselves needed more powerful control over the robot. This led to Beckman’s distribution of a field-service program called Biotest. Originally intended only for testing and calibration of the robot by qualified field-service representatives, Biotest was inadequate for the types of sophisticated programming we needed in the lab. It did, however, give complete control of the robot at the lowest possible level and also retained a few of the higher-level commands of Genesis. Another limitation we found with both Genesis and Biotest was an inability to control other external devices in concert with the Biomek. Aside from this, the major drawbacks remaining with Biotest all lie in areas at which Tcl excels: variables, looping, conditionals, and general structured programming.

Starting from the Biotest code, we were able to integrate the low-level functions of the Biomek into a command-line Tcl interface. Figure 2 shows examples of code written for all three interfaces.

2.2 How is Tcl implemented?

In this implementation the tcltest interpreter was used with only minor modifications for initialization. John Martin's DOS port of Tcl V6.2 (available by anonymous ftp from `cajal.uoregon.edu`) was used. 12 new built-in commands were added to allow complete control of the robotic system (both the Biomek 1000 and the SL). They are: `move`, `home`, `stat`, `config`, `connect`, `disconnect`, `get`, `unget`, `pipette`, `pump`, `vac`, and `tip_rack`. Some of these commands are redundant. For instance, `pipette` may actually be done (and is done, internally) by issuing a `move` command to the pipette motor, but having a separate command enhances readability and also allows for low-level error detection and consistency checks (e.g., is a tool in place to pipette with?).

The problem of connecting to other devices via SYGI was addressed by simply writing external programs to drive these devices with DOS command-line interfaces and then using Tcl's `exec` command with arguments appropriate to the external program. Variables could then be passed to these programs from within SYGI.

Once the hard-coding was done, several SYGI utility procedures were written to facilitate the port from Genesis and Biotest. These capitalized on the benefits of Tcl because the conversion programs themselves could be written as scripts. The users of the robot then quickly established a small library of routines which they could modify and then use repeatedly. Here again, the flexibility of a procedural language was key, since general routines could be written and input parameters changed for each run. Thus, one entire class of biological protocols can be made into a single procedure having appropriate arguments.

2.3 Use of SYGI in Production Sequencing

Once most of the basic procedures are defined, they are run in production mode unaltered. We currently have 10 such procedures, several of which are run daily. To simplify the interface even more, SYGI was created to accept command-line arguments specifying a Tcl script to be sourced and executed on startup (similar to "wish -f" on UNIX). One of the lab technicians added a Pascal interface which allows users to select a sequence of SYGI procedures from a list and then prompts for any necessary parameters. Figure 2 shows one possible building-block procedure for some of these protocols.

Currently, SYGI is being used in at least seven genetics labs, both commercial and university, and has received very high praise from all. Given this response to a program with a crude user interface and relatively little support, Tcl appears to be well suited to this environment.

3 Future Uses for Tcl/Tk in the Genome lab

Having production experience using Tcl in our lab, we are looking forward to other ways that Tcl and Tk could make our lives easier. The rest of this section describes some of our conceptions of future projects.

3.1 Other Robots

One of the natural next steps is to use Tcl to drive the other robots we are developing in house. Clearly, this would make the lives of the staff biologists much easier; once they have learned the language which controls one of the robots, they already know 95% of that which controls the others.

In particular, we are considering driving a new robotic colony picker. This machine (currently run by a Sun SPARCstation IPC) works by digitizing an image of a petri dish with viral colonies grown on it. The driver then interfaces with outside image analysis software to decide where on the dish the plaques are located. Then, the robot positions the petri dish by moving the stage which supports to the appropriate spot underneath a wheel of tungsten picking elements. The plaques are then picked and placed in a 96-well plate on a second stage.

The software development cycle for this device could be accelerated by using a Tcl-based interpreter which could then be left in place for the production use of the machine. Implementing Tcl commands to control and calibrate the stages, the loaders, the digitizer, and the wheel of picking elements could provide an environment which allows for fast development and modification of complex scripts.

An oligonucleotide-synthesizer robot is currently being built around a 486 machine which could benefit in similar ways from a Tcl-based controlling language.

3.2 Control of Multiple Instruments from a Tk Interface

The production setup in the Genome Project may, in the future, have up to five or more robots running at any given time. Once this becomes a reality, it will be especially important to query the status of all of the machines without having to visually inspect each one. It would be even more useful if one could start a run, modify parameters, or otherwise individually control these robots from a single control panel.

This is an application well-suited to Tk. However, as one of our project directors likes to note, "We have a veritable Tower of Babel of computers that need to talk to each other." Specifically, we have robots run by Macintoshes, DOS PCs, and UNIX machines under the control of a variety of driver programs. For those machines whose drivers are not Tcl-based, a simple Tcl-interpreter front-end to the real driver can be written. In implementing this, therefore, we will probably take one of the publicly available `send` commands which works over TCP/IP (e.g. tcpConnect) independent of Tk and port it to the Macs and PCs.

A centralized control panel written in Tk using this separate `send` command should come together fairly quickly. With an appropriate GUI, it would be useful to operators without any knowledge of Tcl. One could start or stop the robots, monitor status, set or tune parameters, etc. For the more advanced user, segments of Tcl code could be entered and sent directly to the robot drivers in order to modify their behavior.

3.3 Sample Tracking & Data Flow

Examining the Yeast Genome Project from a macro level, one can easily see where the aforementioned vast amount of data comes from. Once a plaque has been picked and put into one the 96-well plates, it becomes a named entity. From there, it needs to be tracked (as a physical sample with a logical name) as it works its way through the various technicians, processes, and robots until it ends up as sequence data on our UNIX machines. At that point, we need to track both the sequence data as well as the physical sample of that data.

In the biological community, Sybase is something of a *de facto* standard for database work. In order to make for easy exchange of our data with other scientists, we will very likely choose Sybase as the basis of our sample-tracking software. The existence of Tom Poindexter's SybTcl extension should further solidify our choice of both Tcl/Tk and Sybase.

We will be able to start the life of a sample by having the plaque-picker generate a name for it as it is picked. This name can then be sent to the Tk controller application described earlier, which will pass it along via SybTcl to the sample-tracking database. For the rest of it's lifetime, the sample can then be tracked (plate location, refrigerator, status, point in the process line, etc.).

Furthermore, with the use of bar-coded plates, the process can be sanity-checked during the sequencing process: are we really sequencing the plate and samples that we think we are? Another advantage is that it can simplify manual error correction. For instance, if it is decided that one sample was contaminated along the way and needs to be resequenced, we can find that sample and reinsert it in the middle of the assembly line. Then, the Tk controlling software can be instructed that, on that particular plate, only a single sample is to be extracted and sequenced (unlike the rest, where the entire plate of 96 samples will be processed). The Tk controller will then be able to pass along this information to all the robots and the sample-tracking database, which will proceed to update the database and notify the correct personnel as needed.

4 Conclusions

Based on our current experience with Tcl and a few minor tools written in Tk, we anticipate greater use of Tcl throughout laboratory operations. These may take the form of device drivers, process monitors, and task managers. It is hoped that Tcl may become more of a standard driver for many industrial devices, perhaps even being burned into EPROMs of microcontrollers. Other extensions of Tk are obvious as well, particularly porting to other environments (such as Macintosh and MS-DOS systems) from which devices are already driven. The `send` functionality and modularity of both Tcl and Tk make them obvious choices for process monitoring and task management. Generally, Tcl/Tk seems to be a long awaited product for those developing instrumentation and integrating automated processes. Its use in these and many areas beyond computer science should be encouraged.

5 Acknowledgements

Thanks are due to John Mulligan for vision in automation, to Andy Kayser for testing and new interfaces to SYGI, and to George Hartzell for introducing the lab to Tcl/Tk.

6 Author Information

Scott P. Hunicke-Smith is an Engineer-in-Residence with the Stanford Yeast Genome Project. He is also currently working on his Ph.D. in the Mechanical Engineering Department at Stanford. His e-mail address is `ssmith@genome.Stanford.EDU`.

Dan Mosedale is the Genome Project's Systems Administrator. He received his B.S. in Computer Science from the University of Oregon and has been involved with UNIX system administration and programming for a number of years. He can be reached via e-mail at `mosedale@genome.Stanford.EDU`.