Married with Tcl

J. Schönwälder
Computer Science Department
Technical University Braunschweig
Bültenweg 74/75
38106 Braunschweig
Germany

Abstract

Writing and maintaining a Tcl C extension for several years is an experience which can be compared to a long lasting relationship between a man and a woman. At the beginning, there is great excitement and you enjoy a real good time. However, once the relationship settles, you will find out that being married is sometimes straining and you start to look at things from a slightly different perspective.

This paper documents some of the experiences and lessons learned during seven years of writing and maintaining the scotty package. The scotty package contains the Tnm network management extension for Tcl, which is used at many sites world-wide to automate network management tasks or to drive test-suites for network devices. This paper is an experience report and does not introduce new concepts to enhance Tcl or Tnm. However, some of the issues raised in this paper may prevent others from repeating some of the mistakes made by the author. Experience reports like this one may also help others to make realistic estimations about the amount of effort needed to keep an extension alive.

1 Falling in Love with Tcl

The whole story began in 1991 when I started to work in the area of network management. It soon became clear to me that the software which was available at that time was lacking flexibility. I always felt that commercial tools were in many cases more complex than needed to solve a given problem. On the other hand, freely available command line tools were often a bit too inefficient for the implementation of customized solutions that perform well in a network of reasonable size. In fact, efficient network management requires to do many things in parallel and thus it is important that APIs support an event-driven programming style.

My UNIX system administration background told me that most network management tasks are rather simple. All you need is a relatively small set of customizable scripts to solve at least 80 % of the problems in an operational network. Unfortunately, the tools that were around in the early 90's were not very efficient and smart - so using them in scripts was not performing well.

I started to play with the idea to create my own little domain specific language for automating network management tasks. One day, I stumbled across Tcl on one of the first free BSD releases I was playing with[1]. I obtained the classic Usenix paper on Tcl [1] and soon realized that Tcl is a good candidate for building a powerful network management scripting environment.

Once I started to play with Tcl, I soon learned about Tk [2] – which was the best free X11 toolkit available at that time. I immediately fell in love, especially since it was so easy to learn Tcl and Tk[2]. The documentation was well written and complete. In addition, diving into the C source code was easy since the readability of the sources was outstanding. So I finally decided to write a Tcl extension for solving network management problems, which later was called scotty[3] [4].

I have not been the only one who recognized that Tcl fits nice into this problem domain, nor have I been the first one. There are many commercial products in this area today which support Tcl. The Tnm extension however became the most successful free Tcl extension in this area. I can only speculate about the reasons for its success, but I think that the Tkined application which comes with the scotty distribution and the support on the mailing list were two key ingredients. Readers who want to learn more about the usage of Tcl to automate network management tasks are referred to [5], [4] and [6].

This paper is an experience report and does not introduce any new Tcl features nor does it report on clever ways to use Tcl or future features of the scotty package. A similar experience report has been written by Don Libes for his Expect package in 1997 [7] and this paper shares some of the spirit with Don's paper. This experience report is in many places biased by the opinion of its author. Hence, the author does not expect that everyone agrees to every statement made below. In fact, the paper may lead to some controversial discussions – which is a welcome side effect of writing this paper.

2 Being Married

The beginning of my relationship with Tcl/Tk was very enthusiastic. I enjoyed the time I spend writing the C extension since I learned a lot from the quality of the Tcl C source code. In fact, I sometimes wondered why the Tcl scripts that came with the distribution did not show the same elegance of the C code[4].

Several times, I touched the boundaries of what Tcl provided. One important example was the event loop which I used in a somewhat different way. In fact, the scotty package had to provide its own event loop until the Tcl developers finally decided to move the event loop from the Tk extension into the Tcl core in Tcl 7.5. And even after the integration of the event loop into Tcl, I had to debug the event loop several times since it did not work

[1]My first contact was actually with TclX, which I found a somewhat strange thing. I never understood why you need a help command when you have good man pages.

[2]We used the Interviews toolkit [3] at that time to implement various GUIs. The time needed to learn and use Interviews efficiently was so high that many students spent more time trying to understand Interviews internals than solving actual problems.

[3]`<URL:http://www.cs.utwente.nl/~schoenw/scotty/>`

[4]I later realized that it is much harder to write elegant Tcl code than elegant C code, which is already hard.

correctly for the things I wanted to do with it. The good news is that it was always easy to get into contact with core Tcl/Tk developers. John Ousterhout, the creator of Tcl/Tk, was always very open to good arguments that were backed up by concrete patches.

A new era began when Tcl was moved to Sun Microsystems in 1994. The Web was just beginning to spread in the commercial world and everything had to be Web enabled. Of course, Tcl (like any other language at that time) had to run in a Web browser and this brought us the integration of multiple interpreters and the safe Tcl mechanism into the Tcl core. The idea behind the padded cell approach [8] was nice and the implementation elegant. I actually made use of safe Tcl interpreters by allowing Tcl enabled management agents to execute scripts from untrusted sources in a safe interpreter. I soon learned that safe Tcl/Tk interpreters are pretty useless for solving real world problems and that the creation of useful security profiles is a complex time consuming (and often boring) task. It was also necessary to analyze the C code of the Tnm extension for any constructions that may have a negative impact on security. Of special importance were any constructions where an internal state was shared between multiple interpreters since these internals could be explored by a malicious script in a safe interpreter to attack the trusted interpreter. In many cases, the overall result of this exercise was a cleaner implementation of the Tnm extensions. But compared to the number of scripts that actually use safe interpreters today, the effort spent on making Tnm work with safe interpreters can hardly be justified.

To be really sexy, Tcl/Tk had to run on different operating system platforms. The decision to make Tcl and Tk portable across platforms (rather than making the various platforms look like Unix and X11) caused major changes to the Tcl C API. A good example is the introduction of a complete new I/O system - which was only needed to support non Unix platforms and which had significant impact on all platforms, including Unix. Since I expected that people would ask sooner or later for a scotty version which runs on Win32 systems, I went ahead and modified the package to use the new Tcl I/O system instead of the standard I/O functions of the C library. But I also started to ask myself seriously at this point where this evolution leads to. The little glue language I fall in love with was now providing replacements for standard C library functionality. Not only is the package becoming bigger and bigger, it also gets harder and harder for newcomers to understand all the pieces. And even more important: The glue language started to dictate how to write applications or extensions by putting constraints on the usage of the standard C library.

It did not take long until people asked for a scotty version which runs on Win32 systems; systems I never really programmed before. Being a nice husband, I bought a C compiler[5] and started hacking. Following the Tcl style, I moved all the platform dependent code into separate C modules, which of course required to introduce or redesign internal interfaces. I finally managed to get things running reasonably well - but debugging and maintaining the Win32 version still causes me headaches.

In 1998, Scriptics was founded, which increased the pressure on the core developers to make money with Tcl/Tk. This led to some very interesting additions like thread support, even though the father of Tcl is known to prefer event-driven programming over threaded

[5] I actually started with the Borland compiler until I learned that choosing a compiler means something very different in the Win32 world compared to the Unix world. I later switched to Visual C++ just before support for Borland was dropped by the Tcl/Tk core.

programming[6]. In order to support threads, it is necessary to analyze the C code for constructions that need to be protected by mutexes. Manual code inspection is time consuming, boring and error-prone. Unfortunately, it is hard to verify that a piece of code correctly uses all the mutexes. There is a high risk that you have missed a few places, which may show up as unpredicted behavior of a program or even sporadic segmentation faults (sometimes with garbled stacks). The interesting thing is that nobody on the mailing list[7] for the scotty package ever asked seriously for thread support. The event-driven APIs for the network management protocols seemed to work fine, especially since there is no real benefit from computational parallelism when managing networks.

Other additions made the core Tcl interpreter more efficient, most notably the introduction of the byte compiler and Tcl_Objs in Tcl 8.0. As a side effect, Tcl was now in principle able to deal with 0x00 bytes in strings, which is very valuable for a Tcl extension that provides networking support. However, in order to take full advantage of Tcl_Objs, one needs to rewrite some parts of the C code. Furthermore, it is necessary to do proper reference counting, which obviously complicates the process to find and eliminate memory leaks. It is thus not too surprising to me that the Tk extension has not been converted to use Tcl_Objs internally everywhere.

Namespaces and packages are other examples of very valuable added features. The impact of namespaces on the Tnm C extensions were minimal. The real effort here was the conversion of scripts that are part of the distribution to use namespaces. Unfortunately, Tcl namespaces have the property that they do not protect data against (un)intended modifications. It is thus hard to write very robust libraries of Tcl code since you can be sure that some people will play dirty games with internal data structures. Namespaces are thus only a tool to better organize the growing number of Tcl commands and to resolve name clashes. A feature which is usually used in conjunction with the namespace system is the package mechanism. The default behavior of the `package require` command was to delay the actual loading of the package until the first command of the package is called. This behavior leads to some strange side effects, e.g. a command in an extension which usually can never fail might suddenly throw an error since the loading process for the package failed. Only few Tcl programmers will be able to deal correctly with these kinds of delayed package loading errors[8]. Fortunately, it was possible to write hand-crafted scripts to load packages directly during a `package require` invocation and it seems that Tcl is now on its way to make this behavior the default.

Internationalization brought us support for UTF8 [9] and Tcl_UniChar. So far, I did not dive into the details and implications of these recent additions, probably because the Latin-1 character set causes no real problems for Europeans. Hence, the immediate need to worry about international character sets has not been big enough to really explore this issue.

[6] John Ousterhout gave an invited talk at the 1996 USENIX Technical Conference which was titled ”Why Threads Are A Bad Idea (for most purposes)”.

[7] `<URL:mailto:tkined@ibr.cs.tu-bs.de>`

[8] I complained about the behavior of the `pkg_mkIndex` command shortly after the introduction of the package mechanism but failed to convince people that this is a problem.

3 Unfulfilled Expectations

During my relationship with Tcl, I developed several expectations on my mate. Some of them were satisfied while others still exist as unfulfilled expectations.

The Tk toolkit was for many people a strong argument to use Tcl in the early 90's. Other scripting languages such as Perl or Python were actually embedding Tcl just to get hold of the Tk toolkit. Many of the programmers using the Tk toolkit shared the expectation that the toolkit will evolve and support additional widgets. However, only little progress was made in this area and the appearance of the core Tk toolkit looks a bit old fashioned today. Students who would have been using Tk (and thus Tcl) a few years ago are now using other toolkits, such as GTK+ [10] or Qt [11]. Perhaps the Tcl/Tk developers did not really recognize the importance of the Tk toolkit for the acceptance of Tcl itself since they have focussed too much on the evolution of the core Tcl interpreter.

But even though a lot of energy has been spent on the Tcl core, I must notice that the Tcl language did not really improve that much. In fact, I am in the group of people who claim that the consistency of the language got worse over time. There seem to be no common rules for syntactic elements and there is no clear plan in place on how to migrate old command interfaces to new ones. A frequently named example is the `string` command which provides many subcommands to process strings while there are many list commands that operate on Tcl lists. Similarly, it seems more natural to use an object-based command interface for open channels. Additions such as the `fconfigure` command probably indicate that the Tcl developers do not care enough about the namespace they are responsible for. Another indicator is the fact that many utility procedures and variables still do not use namespaces and instead live in the root namespace. Of course, backwards compatibility is an important issue. But there must also be a path to evolve the language and to overcome mistakes in order to keep the whole language comprehensible and consistent. Today, the Tcl language has more complexity than actually needed.

Another expectation grew when dynamic loading was introduced. I expected that this is a milestone which will evolve Tcl to a more modular language core where many language features live in dynamic loadable packages. People who only need a tiny interpreter without any extra features would then be able to keep the price for embedding Tcl at a minimum. A more modular Tcl core would have helped to strengthen the package system. As a secondary effect, the life for extension writers would have become easier since they could borrow ideas for packaging dynamic loadable extensions from the Tcl core. I believe that more aggressive use of dynamic loading in the core could make efforts such as the Tcl Extension Architecture (TEA) [12] mostly superfluous since all information needed would be part of the core itself.

4 Falling in Love Again?

Perhaps it is a good point in time now to ask myself whether I would fall in love again with Tcl today. I still believe that network management is an area where scripting facilities play a key role. So I still believe in the core ideas of the scotty package and I would again start with an existing language rather than inventing my own specialized domain specific language.

Regarding the choice of the language, I would take a serious look at Python [13]. I believe that Python makes it easier to write elegant and maintainable code. Python has object orientation while Tcl still struggles to integrate [incr Tcl] into the core. Furthermore, Python comes with a very large library for many things that I consider useful for network management scripting. The Tcl developers just started to provide such a standard library, which is unfortunately restricted to Tcl code only for portability reasons. It may take years until Tcl reaches the same functionality already present on most Python installations. I would also take a critical look at the event handling mechanisms of potential scripting languages. This is an area where Tcl has a good chance to outperform its competitors.

For writing graphical user interfaces, I would take a serious look at GTK+ [10]. It has a nice look and feel and it seems to evolve at a reasonable speed. There are already bindings to many programming and scripting languages, including Python[9].

Taking the pieces together, I would probably not start again with Tcl/Tk if I were to start from scratch today. There are clear competitors out there and I would take a serious look at some of them. But this does not mean that I am now going to divorce myself from Tcl/Tk. Seven years are a pretty long time and I need much stronger reasons to break a relationship that survived so many ups and downs.

5 Lessons Learned

My marriage with Tcl had of course an impact on myself. I learned many things and I probably changed my minds several times due to my experiences and the world changing around me.

First, I have learned a lot about C programming from reading the Tcl C source code (when it was still simple to understand). Reading the source code teached me a lot about good programming style, even though I may not agree with all the little bits and pieces today. I also enjoyed the experience of how powerful a few lines of C code can get if you embed the code into a general purpose interpreter. Being able to invent new control structures (e.g. domain specific loop constructs) is a very powerful mechanism to make scripts short and elegant. The ability to prototype in an interpreted language and to recode in C for efficiency is very valuable in environments where the time to realize an idea is small and the number of programmers is limited.

The second important lesson I learned is the importance of separating the scripting API implementation from the implementation of the functionality itself. In other words, it is very valuable to design a clean C API for the functionality provided by an extension with a thin layer on top of it which provides the interface to the scripting language. Scotty does

[9]There is no GTK+ binding for Tcl yet — but I guess it is just a matter of time until some strange soul writes one.

not follow such a design, which was in retrospect the biggest mistake I made[10]. In fact, we are slowly working on making core Tnm components independent from Tcl. The libsmi[11] MIB parser library is an example of such a component which will replace the Tcl specific MIB parser currently embedded in the Tnm extension. The interesting thing to note here is that Tk suffers from the same problem. Tk was written specifically for Tcl, which made it a bit harder for other languages to adopt Tk as their GUI toolkit. I believe that the impact of Tk would have been different if Tk was designed as an embeddable C library with a scripting API on top of it.

The third lesson learned has to do with software versioning problems. Tcl has very clear rules to distinguish between officially exported symbols and symbols that are for internal use only and which can change at any point in time. The Tcl developers also took great care to use release numbers wisely in order to distinguish between major releases which can cause incompatibilities at the script API level and minor releases which should not introduce major incompatibilities. I once tried to support a version of the scotty package which can be used with multiple major and minor Tcl version. The price for doing this turned out the be extremely high. Current scotty versions are just guaranteed to compile and run only with a specific Tcl minor version because the changes in the C API and the configuration setup make it too time consuming to support multiple versions.

The fourth lesson learned is that cross platform portability is a pain. It obviously makes little sense to try to maintain a port on a platform you are not using yourself. Things are easy as long as things work as expected. But once a real problem shows up, you realize that you lack the knowledge to track the problem efficiently. Despite this non-technical aspect, I expect that in the long term platform differences will become less and less important. I also believe that approaches which try to hide platform differences at the C library layer will eventually succeed in reducing cross platform development and maintenance costs.

The final lesson I learned is that it is hard to make trade-offs between language compatibility and language misfeatures. For example, I picked the %-substitutions idea from the Tk toolkit in order to pass event details to callback scripts since it is the normal way to access event details in Tk. My programming experience with Tk told me already that this substitution mechanism leads to many quoting problems. Nevertheless, I decided to use the %-substitutions trick in Tnm because I expected that programmers will be familiar with it (and its pitfalls). This is obviously not the case. In fact, I know that there are several scripts which will break if you put interesting character combinations in a network management message. In some cases, it is even possible to exploit security holes by carefully constructing messages. I believe that a much better way to pass event details to callback scripts is to use Tcl variables or an object-based Tcl command which represents the event itself. Such a mechanism will also prevent the Tcl runtime system from recompiling the callback script every time it is executed. It would be nice to have a solution which is generally accepted and which will at some time even replace the current handling of Tk events.

[10]The original reason to use Tcl internals in the implementation of various protocols was speed. It was possible to save some processor cycles by avoiding copying operations into internal data structures. With the speed of todays hardware, the performance benefits became less important in most environments.

[11]`<URL:http://www.ibr.cs.tu-bs.de/projects/libsmi/>`

6 Conclusions

This paper took a critical look at the evolution of Tcl/Tk from the perspective of an extension writer and maintainer. There are many more things that could be written down in this context. The selection of the issues and my comments on various aspects of the Tcl/Tk toolkit are certainly not objective. Even if the paper sometimes highlights more negative aspects than positive ones, I must notice that the overall scotty project was a success and that I did also have a lot of fun during the last seven years with Tcl.

Acknowledgments

Thanks to Frank Strauß for reviewing this paper and making valuable suggestions.

References

[1] J. K. Ousterhout. TCL: An Embeddable Command Language. In *Proc. Winter USENIX Conference*, pages 133–146, 1990.

[2] J. K. Ousterhout. An X11 Toolkit Based on the TCL Language. In *Proc. Winter USENIX Conference*, pages 105–115, 1991.

[3] M. A. Linton, P. R. Calder, and J. M. Vlissides. *The Interviews User Interface Toolkit*. Prentice Hall, 1994.

[4] J. Schönwälder and H. Langendörfer. Tcl Extensions for Network Management Applications. In *Proc. 3rd Tcl/Tk Workshop*, pages 279–288, Toronto (Canada), July 1995.

[5] M. T. Rose and K. McCloghrie. *How to Manage Your Network Using SNMP*. Prentice Hall, 1995.

[6] D. Zeltserman and G. Puoplo. *Building Network Management Tools with Tcl/Tk*. Prentice Hall, 1998.

[7] D. Libes. Writing a Tcl Extension in Only 7 Years. In *Proc. 5th Tcl/Tk Workshop*, pages 55–65, Toronto (Canada), July 1997.

[8] J. Y. Levy, L. Demailly, J. K. Ousterhout, and B. Welch. The Safe-Tcl Security Model. In *Proc. USENIX Annual Technical Conference*, June 1998.

[9] F. Yergeau. UTF-8, a transformation format of ISO 10646. RFC 2279, Alis Technologies, January 1998.

[10] H. Pennington. *GTK+/Gnome Application Development*. New Riders Publishing, August 1999.

[11] M. K. Dalheimer. *Programming With Qt.* O'Reilly & Associates, Inc., May 1999.

[12] B. Welch. The Tcl Extension Architecture (TEA). In *Proc. 7th Tcl/Tk Conference*, February 2000.

[13] D. M. Beazley and G. van Rossum. *Python Essential Reference*. New Riders Publishing, October 1999.