The following paper was originally published in the
Proceedings of the Fifth Annual Tcl/Tk Workshop
Boston, Massachusetts, July 1997

Tcl in AltaVista Forum

David Griffin
AltaVista Internet Software Inc.

Tcl in AltaVista Forum

David Griffin
AltaVista Internet Software Inc.
dave.griffin@altavista-software.com

Abstract

AltaVista Forum™ is an award-winning collaboration environment based on the open technologies of the World-Wide Web and built on the foundation of the Tcl language [pob]. Using Tcl's inherent extensibility the AltaVista Forum toolkit provides a simple class/inheritance mechanism, an information manager customized for the data storage needs of collaboration applications, and a growing toolkit for creating asynchronous collaboration applications on the Web.

This paper details how Tcl has been employed as the basis of a commercial web-based collaboration environment. While the focus of the paper will be on the design and our use of Tcl in AltaVista Forum 98, we will also touch on how the design has evolved over the past three releases of the product.

Selecting the Tcl Language

AltaVista Forum (formerly Workgroup Web Forum) is described pretty thoroughly from a functional standpoint in [Chiu]. If all has gone well, we will have released the third version of this product since its debut in 1995: spanning the use of Tcl 7.3 through 7.6 (and looking forward to Tcl 8). (More information can be found at: http://www.altavista.software.digital.com/forum)

In late 1994 a small team of engineers in Digital's Networks group began pursuing two goals: building products which worked within the framework of the then emerging World-Wide Web and building them in a new way which matched the rapid pace of evolution expected in this space.

Rapid development, flexibility, and platform independence pointed us towards interpreted languages as a core technology for constructing these new products. Perl, Python, Tcl, and a number of other interpreters were examined and evaluated. Tcl emerged as the tool of choice and in a matter of weeks a toolbox of extensions began to come together along with a flurry of prototypes.

The selection of Tcl as a core technology came about as a compromise based on a variety of factors and, to be sure, a bit of cultural pressure within Digital. For example, Perl was arguably more popular and therefore would be more attractive to people who wanted to program in our environment. Python had a very attractive object model and reasonably clean extension mechanism.

Tcl was selected because it was a mature (version 7 and still cooking), portable (most UNIX® platforms, Windows®, and Macintosh®), and highly extensible language that possessed a vibrant user community (comp.lang.tcl), an excellent book available in most bookstores [Ousterhout], and (as luck would have it) a commitment of support by a respected engineering company (Sun Microsystems). We've been generally happy with our choice.

Initial Design Elements

Following in the footsteps of Tk (and others) we created the concept of the AltaVista Forum "Toolkit": a custom scripting language for developing a class of web-based collaborative applications. Using Tcl as the core language we extended it with both public domain and privately developed code. The goal was to develop a language that was sufficiently powerful so that simple collaborative applications could be scripted in just a few pages, but would also allow for the development of sophisticated and customizable applications as well. We also wanted to provide a language which would execute identically on multiple target platforms (specifically UNIX and Windows NT™) while not excluding the ability to use Tcl to its fullest extent (we don't disable any commands, including `exec`),

Data Management

The ndbm library from Berkeley formed the base of our database manager. We constructed a simple Tcl-object interface for it and ported it to Windows NT. (In our third release we also added a simple journalling and recovery facility to improve reliability).

Simple key/value data management was too low-level for the applications we were envisioning, so we created an "information manager" which was layered on top of the ndbm object. Initially written entirely in Tcl, this provided for hierarchically structured data access with named fields (attributes) which could be dynamically modified without management intervention (to match the dynamic nature of an interpreted environment), along with a general purpose "properties" facility. This was the basis for our "structured" data management, with the file system holding our unstructured data (the "blobs" of information that were the heart of the content we were managing).

Critical to our vision of collaborative data management was augmenting the database with a content retrieval mechanism: a search engine. We initially designed a set of Tcl commands which would allow multiple search engines to be used. For our "out of the box" product we decided to use a Digital-developed search engine called "NI2" - which would later become the core of the AltaVista Search engine. While the interface for using multiple search engines still exists, the capabilities of the NI2 facility are so heavily used that it would be difficult to integrate another search engine.

Web Tools for Web Apps

Because we were a web-based application, we needed to both interact with hypertext servers as "CGI" applications, and generate results using the HTML language. Glenn Trewitt of Digital's Network Systems Lab had released a paper on using Tcl for HTML forms processing [Trewitt] along with some C libraries. We took this code, ported it to Windows NT, and added other features that we required. In parallel, the "HTML library" took form as a collection of Tcl commands and objects which attempted to abstract the various parts of HTML page generation.

The Toolbox

Rounding out the toolkit was a suite of commands that we felt made it easier to write compact and portable scripts.

The platform command set provided for platform-independent file management (copy, delete, rename). Another set of commands managed internal date/time stamps. (Both of these facilities have been partly or completely replicated in recent releases of the Tcl core).

AltaVista Forum operates in several languages (English, French, German, Spanish, and Japanese). The toolkit provides the ability to internationalize applications using "native language tags" embedded in the code and to build catalogs that can be translated later.

A few low-level programming constructs were added as well:

- The `try-else` command is a handy abstraction for Tcl's `catch` command which has become rather popular with our developers.
- The `isnull` and `strequal` commands are shorthands for the Tcl `string compare` command. Tcl's relational expressions can be tricked into raising errors when presented with long strings of digits.
- A "mailbox" object employs a simple SMTP mail client allowing applications to generate mail messages or to read mail messages from POP servers.

Applications, Classes, and Inheritance

The design of the toolkit attempted to encompass the ideas of objects, classes, and inheritance. incr tcl was attractive as a Tcl object framework, but we were concerned that anything which required modifications to the Tcl core might put our product at risk for Tcl patches or releases – so we opted for our own, very simple, set of object-based mechanisms.

AltaVista Forum applications are built from a set of class files and module files. Module file declarations are nothing more than glorified `source` commands. Modules were added after our applications grew to be much larger than originally envisioned and needed to be broken down into more manageable chunks. Class files are similar to modules, but instead of simply sourcing them in, their inclusion in the application is via the `forum inherit` command, which applies the toolkit's inheritance semantics.

At the outermost level an AltaVista Forum application consists of a "class file": a series of `forum` commands which declare different types of objects. These class files can also inherit other class files, allowing redefinition of the objects. This mechanism allows for customization of applications without directly modifying the applications supplied in our "out of the box" product.

The forum objects consist of an amalgam of global variables, global arrays, and procedures/commands which follow an internal naming scheme, thereby creating separate namespaces which (ideally) could coexist in a single interpreter. The more practical aspect of this mechanism is that AltaVista Forum application developers could pretty much program freely in Tcl and not worry about bumping into the toolkit infrastructure which was easily manipulated through toolkit commands. For example, the command: `forum button b1 -text "b1 text" -image b1.gif -mref b1Message` ends up as entries in the class-specific array for buttons. When the button is used later in the context of a toolbar, the object name b1 is handed to a processing routine which can determine the current class name, which determines the global array name.

The following excerpts of an AltaVista Forum application demonstrate the syntax and some of the features of the language.

```
forum class paper_demo "Tcl Paper Demo"
#
# Sample of language syntax.  This
# doesn't really do anything useful.
#

#
# Bring in standard support code
#
forum inherit _stdwgw
forum inherit _acl
forum module utiities
forum module event

#
# Extended database with additional
# attributes (fields)
#
forum attribute editStatus {
   {type text} {default "open"}
}
forum attribute allowEdit {
    {type text} {default no}
}

#
# Define some HTML form composition
# elements.
#
forum textfield t.title -mapto title \
  -cols 60 -wrap virtual \
  -label "Title" -labelid l/title
forum checkbox c.allow -mapto allowEdit \
  -label "Allow edits" -labelid l/allow \
  -labelafter

#
# Sample toolbar button and toolbar
# definitions.
#
forum button tb.showDoc -text "Show" \
  -mref "showDoc %3" -senseid 3 \
  -image tb_showdoc.gif \
  -inactiveimage tb_showdoc_grey.gif

forum toolbar mainbar {
  std.home tb.showDoc
}

#
# Sample message that lists the titles
# of child documents, with hypertext
# links to those documents by clicking
# on the title.
#
forum message listSubDocs {
    # Open database.  Do access check
    # acl_deny_dialog is inherited from
    # the _acl class.
    wim open r
    if {![acl_permits]} {
       forum_exec view acl_deny_dialog
       wim close
       return
    }
    # Parent document is passed in the
    # message arguments (from URL)
    set docId [lindex $margs 1]
    begin_response text/html normal
    emitln [html head] [title_bar]
    emitln [html body] [std_header]
    emitln [toolbar mainbar $dodId]
    emitln <h1> [nlt h/lsd1 \
       "Subdocumment List for"] \
       [aval title $docId] </h1>
    set count 0
    # Enumerate children.
    foreach_entry -childof $docId d {
        incr count
        emitln [mlink [aval title] \
             showDoc $d]  "<br>"
    }
    emitln "<p>"
    emitln [format [nlt h/lsd2 \
      "Total subdocuments: %d"] $count]
    emitln [std_trailer]
}
```

Executables

The AltaVista Forum system is controlled by three executables: the dispatcher, the butler, and the background service. While the applications are shipped as files that can be modified by customers, the toolkit is a bit more protected. The hybrid C/Tcl code was merged together with a custom derivative of the Embedded-Tk [Hipp] processor which, as the name implies, allows Tcl code to be embedded into a C program framework.

The dispatcher is the web server CGI application where most of the work in AltaVista Forum is performed. Web browsers (such as Netscape Navigator™ or Microsoft Internet Explorer) contact a hypertext server, which in turn creates a process and runs the dispatcher. The dispatcher (a Tcl interpreter extended by our toolkit with a fixed initialization script) analyzes the request, loads the application, executes the appropriate scripts which ultimately generate an HTML response which is transmitted to the web browser for formatting. The dispatcher then runs down and the process is deleted.

The butler is the AltaVista Forum's version of `tclsh`. It has most of the same extensions that the dispatcher has (minus the elements only of use to a CGI program), and includes a number of forum management commands, some application development aids, and a reasonable reproduction of the dispatcher execution environment suited to offline processing.

The background service is a program that coordinates the execution of tasks that are to be performed on a particular schedule or as a reaction to some event in the applications. The background process doesn't actually do any of the work but instead exec's butlers with small scripts.

Code Volumes

The table below chronicles the growth of the toolkit proper (this does *not* count the code in various components we integrate into the toolkit such as Tcl, ndbm, NI2, etc.)

	C		Tcl	
Release	# Files	Lines	# Files	Lines
1.0	15	5,435	19	9,039
1.0A	15	7,338	19	8,900
2.0	15	9,286	19	12,994
2.0A	15	10,027	19	13,505
3.0	32	18,829	28	14,883

Compared to the applications, the use of Tcl in the toolkit has grown only modestly. The numbers reveal a bit of the toolkit team's development strategy: where possible first implement the functions needed in Tcl and then recode in C to gain performance after the interfaces and behavior have stabilized

The table below shows the growth in the forum applications code volume (essentially lines of Tcl). The ten-fold increase in size is due in part to new applications, a large number of new features being implemented, and a wider range of engineer experience with Tcl. It also reflects the toolkit lagging behind the applications (there are more application engineers than toolkit engineers) which is made up in lots of Tcl code. With a little luck, this number might actually start to decrease in future releases as we continue to fine-tune the toolkit.

.RELEASE	# FILES	LINES
1.0	8	6,700
1.0A	8	17,104
2.0	30	48,398
2.0A	34	56,773
3.0	114	97,591

Latest Design Elements

The design presented thus far served the first two major releases of the AltaVista Forum product quite well, but suffered from a performance problem that limited the scalability of the product. In our third release we were allowed to attack this problem rather aggressively. The resulting design utilized the Tcl core to an even higher degree.

Tcl Package Libraries

The first change was to replace the Embedded-Tk mechanism with a new facility called Tcl Package Libraries (TPL). All of the toolkit Tcl code is now placed in a single archive which is read in by the various executables - bringing in the particular modules they require. The TPL mechanism serves a number of purposes:

- It reduces our kit size because the Tcl code isn't replicated between 3+ programs.

- It provides a vehicle for extensive customization by resellers – an attractive feature for our product.

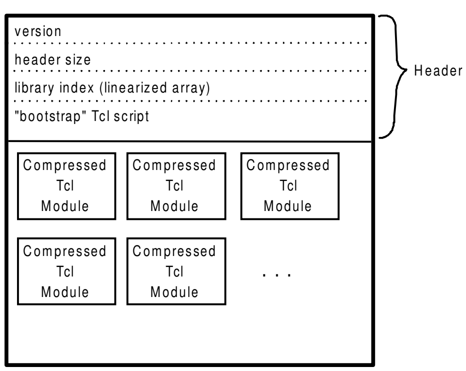
- It could potentially be encrypted to hide sensitive code (we don't do this in our product).

- We anticipate that Tcl source modules could be replaced by Tcl8 bytecodes [Lewis], with commensurate performance improvements by avoiding the on-the-fly compilation phase.

The TPL archive is created by a Tcl script which reads in the designated Tcl modules, compresses the comments out, and arranges them in a single file with an index structure in the header area. The archive script can also write out a "bootstrap" script into the archive which is a small script that locates and reads in the more robust package_load facility. This keeps the amount of Tcl code that needs to be embedded in the C programs to just a few lines.

TPLs are not connected in any way with "packages" as implmented in the Tcl core. The main benefit of TPLs is that all of our Tcl code in the toolkit is now bound into a single, platform-independent file which is easily distributable over the Web (for patches or incremental extensions).

The diagram below depicts the on-disk layout of the Tcl Package Library:



Structure of the AltaVista Forum Tcl Package Library

AltaVista Forum ships with a TPL containing all of the toolkit. Through configuration files the software can be instructed to load in different modules and/or additional TPLs.

Persistent Server Design

From the earliest prototypes of AltaVista Forum we knew that the high-level design of the dispatcher was nowhere near optimal for high-performance or large-scale deployment. Each transaction required a large amount of processing to occur, sometimes to do relatively little "real" work. The initial focus in release 3.0 was to quantify and then reduce this overhead before pursuing other tuning opportunities.

The first task was to instrument the dispatcher with simple probes that recorded `the [clock clicks]` at each point. A particularly slow CPU was used so that the clock resolution would not mask the results (the actual numbers used in this paper aren't nearly as important as the ratios). Here is a summarized sample of the instrumentation probes that we used as our reference transaction:

Acc. µsecs	Function
44,797	Tcl Core Initialized
52,098	AVF Toolkit (C language) Commands Installed
3,226,111	AVF Toolkit (Tcl language) Procedures Installed
4,742,309	Dispatcher Configuration and Transaction Initialization
12,370,021	Application Class Loaded
12,645,428	Transaction Preparation Complete
15,999,848	Transaction Complete

The timings show that for this 16 second transaction, about 5 seconds was engaged in fixed transaction setup, 7.6 seconds was spent loading the application into the interpreter, with the application spending less than 3.5 seconds actually doing the work (in this case the application was one of our smaller ones and the actual task was extremely simple).

This relatively simple analysis, combined with our knowledge about the growth of AltaVista Forum application sizes indicated that a major performance gain could be achieved if we reduced the time to setup for a transaction. The bulk of the application code, however, also created a constraint: whatever changes were made could not introduce the need for many modifications to the applications (the goal was zero changes required).

After prototyping our own protocol and proving the concept of a persistant server, we elected to use the FastCGI [Brown] interface as the means of interacting with the hypertext server (assuming our now familiar role of being the first to port it to Windows NT). Using FastCGI meant that we could have a process resident in memory waiting for a transaction, process it, and then set up for the next one – avoiding the fixed overhead of toolkit initialization on each CGI transaction. Referring back to our reference transaction, that gets rid of about 4.7 seconds per transaction. Because of the limited deployment of FastCGI we opted to use a CGI to FastCGI bridge program which preserved nearly all of the benefits of FastCGI while permitting us to integrate with a large range of hypertext servers.

The Interpreter Triad

A single interpreter processing these transactions was the first approach prototyped. Routines were created that would save and restore global state between transactions – however we were concerned that applications manipulating various global variables would eventually affect each other as their state was retained in memory.

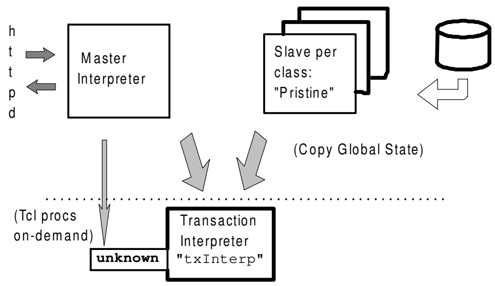
The final design incorporated the use of multiple interpreters configured in a unique way:

- The “master” interpreter holds the initialization state of the dispatcher. This interpreter accepts transactions from the hypertext server via the FastCGI interface.
- The “transaction” interpreter (`txInterp`) is a transient slave interpreter created by the master interpreter for each transaction. Ideally an application executing in this interpreter thinks it is in the same environment as the earlier releases of AltaVista Forum.
- A set of “pristine” slave interpreters are maintained by the master interpreter: one for each application class. The application is loaded into this interpreter, essentially “compiling” the application into its global state variables and procedures.

The job of `txInterp` is to perform the work of the transaction and then disappear - taking any excess global state changes with it. In the reference model it cost 8,700 microseconds for Tcl to create a slave interpreter (with teardown costing about the same) - so this overhead is pretty much lost in the noise.

The pristine interpreters exist to amortize the cost of loading the application classes (7.6 seconds in our reference transaction, and typically much, much higher) across a large number of transactions by loading it once and then keeping it in memory for the life of the process. A typical installation of AltaVista Forum has less than a dozen classes to contend with, so maintaining an interpreter per application class incurs a moderate, but reasonable memory penalty.

Commands were then written to redefine toolkit commands (C extensions) in the slave interpreters, and to quickly copy global variables from one interpreter to another. The typical cost of copying the pristine global variables to the transaction interpreter was approximately 20,000 microseconds.



Tcl Interpreters inside the AltaVista Forum "Dispatcher"

Procedure “Faulting”

Tclsh’s autoloading facility provided the inspiration for the final piece of the new design. Because a transaction typically only uses a fraction of the application procedures (and the toolkit), we knew that avoiding defining all of the procedures for each transaction would save even more time. We used the `unknown` command as a “procedure faulting” mechanism and registered it as an alias in the slave interpreters. When a transaction needs a procedure that doesn’t exist, the `unknown` alias intercepts it and checks the pristine and master interpreters for the procedure. If located, the procedure is transferred to the slave interpreter and executed. The original implementation of this faulter cost approximately 122 microseconds to copy a procedure, but taking advantage of access to Tcl internals this was reduced to 66 microseconds. This meant that for the first time small transactions executed much faster than the more complicated ones.

The procedure faulter is instrumented so that every procedure that flows through it is recorded in a list. We will use this information to determine the most frequently referenced commands which will become top candidates for re-coding in C.

Transaction Flow and Results

The final transaction sequence conceptually looks like this (for performance reasons some things are actually done out of order):

1. Master interpreter accepts the transaction data from the FastCGI interface and parses the incoming message storing relevant information in master interpreter global variables.
2. Create the txInterp; define the (static) toolkit commands.
3. Copy the global variables from the appropriate pristine interpreter. If there is no pristine interpreter resident for the application, then create one and load the class into it.
4. Copy the appropriate global variables from the master interpreter to the txInterp. These variables hold the per-transaction state.
5. Rig the procedure faulter to search the appropriate pristine interpreter and master interpreter for unknown commands.
6. Evaluate the appropriate transaction script in the txInterp. Send response back to the hypertext server.
7. Delete the txInterp. Go back and do it all over again.

This arrangement has proven to be extremely effective. For simple transactions a performance increase of 5X was not uncommon. Large applications developed entirely on the prior version of AltaVista Forum ran with only a handful of changes in some areas that did some operations that were probably better off in the toolkit anyway (most of the changes were related to CGI/FastCGI implementation differences and not to the interpreter design changes).

This mechanism has been essentially replicated in the butler program as well, permitting batch processing of lots of forums (application instances) relatively efficiently.

Observations

When this paper was submitted we had just completed the main development phase of AltaVista Forum 98 (Version 3), so the degree of success of the new design cannot be completely ascertained. However we have learned a few things worth noting:

By carefully isolating the developer from the platform with a wide variety of data management facilities the AltaVista Forum scripting language permitted identical execution on Windows NT, Solaris®, and Digital UNIX™.

While developers generally liked the power of the Tcl scripting language, the lack of a syntax checker and adequate debugging and performance profiling tools was often a source of distress for those accustomed to the more traditional development environments and tools. We are actively working to correct this situation.

The realities of commercial product development meant that we could not fine-tune the toolkit forever. Consequently it is not as powerful and expressive as originally hoped. Tcl's inherent power and flexibility essentially worked against us here: what the AltaVista Forum Toolkit team couldn't provide quickly enough was simply "invented around" by writing more Tcl code. This increases the size of our application code and decreases overall performance.

Our experience with Tcl continues to remain a general pleasure. It is stable and generally does what is documented. We did encounter a few problems worth noting:

- Tcl's C-level routines don't allow access to a lot of important things: like `info` command data. This meant that some our low-level code must still call `Tcl_Eval()` to execute small scripts - partially mitigating performance gains or we were forced to access Tcl internal data structures – which will make migration to future versions of Tcl more expensive.
- Tcl channel I/O and pipe interaction work just differently enough between Windows NT and UNIX to keep us on our toes. One major design feature of the background facility had to be reworked when it was discovered that it didn't work properly on Windows NT.
- We make only one modification to the Tcl core: we stub out Tcl_Init() so that slave interpreters don't need to read init.tcl.

Because a significant percentage of our product is written in Tcl, we can take advantage of this to a great extent in customer support situations. We can construct tools "on the fly" to deal with new problems, and because our configuration files are also Tcl scripts, we can "patch" misbehaving parts of the toolkit in the field without major rebuilds of the product. This feature has proved to be so valuable that we've taken pains to assure that many more parts of the toolkit are now more easily field configurable. This will increase both the supportability of the product and make it more attractive to resellers who wish to make modifications to the toolkit's behavior without necessarily changing code provided by our company.

Acknowledgments

The AltaVista Forum product is the handiwork of more talented engineers at AltaVista Internet Software than I can name here. However I'd like to particularly thank the engineers who were part of my toolkit teams located both in Littleton and Australia that developed the Version 1 and Version 2 releases and laying the foundation for the Version 3 design. Special thanks to Roy Klein, Bob Travis, and Peter Hurley who provide constant and invaluable feedback on one harebrained idea after another. Finally I'd like to thank my original partners in crime, Dah-Ming Chiu and Dave Cecil, who held the early product together, and Larry Augustus for keeping the whole boat afloat.

References

[Chiu] Chiu, Dah Ming and Griffin, David. "Building Collaboration Software for the Internet." Digital Technical Journal, Vol 8. No. 3 1996 http://www.digital.com/info/dtj

[Hipp] Hipp, Richard. "Embedded Tk" http://users.vnet.net/drh/ET.html

[Lewis] Lewis, Brian. "An On-the-fly Bytecode Compiler for Tcl", The Fourth Annual Tcl/Tk Workshop Proceedings, Monterey, California, July 10-13, 1996.

[Brown] Brown, Mark. "FastCGI: A High Performance Gateway Interface", Fifth International World Wide Web Conference, 6 May 1996, Paris France. http://www.fastcgi.com

[Ousterhout] Ousterhout, John. "Tcl and the Tk Toolkit." Addison-Wesley 1994

[pob] DataComm Magazine Hot Products Award, Jan. 1996; PC Week Lab Analyst's Choice, Feb 25, 1996; PC Magazine Editors' Choice April 1996; PC Computing 1996 Finalist MVP

[Trewitt] Trewitt, Glenn. "Using Tcl to Process HTML Forms", Unpublished Digital Network Systems Laboratory Technical Report.