Tcl/Tk-based Agents for Mail and News Notification

DON LIBES

National Institute of Standards and Technology
Bldg 220, Rm A-127, Gaithersburg, MD, 20899, U.S.A.
(email: libes@nist.gov)

SUMMARY

Two agent implementations are described – one for mail notification and one for news notification. Both agents are implemented using Tcl. This paper provides a brief history and perspective of similar agents. Included are experiences using Tcl as an agent-implementation language and comparisons of the results to similar agents. Also described are some new techniques of interest to Tcl programmers.

Keywords: agents; biff; mail notification; Tcl; Tk; Usenet news notification.

INTRODUCTION

Agents are semi-intelligent software systems that autonomously provide service to humans and provide and use services to and from other programs.[1,2] Our experience in engineering domains demonstrate that agents encourage code-reuse and ease the construction of some types of complex systems. However, the lessons of agents are amenable to many domains and applications.

A common type of agent provides notification about new or modified information, such as mail. Two such agent implementations are described – one for mail and one for news – both implemented using Tcl.[3] While notification tools are seemingly prosaic, their very simplicity allows us to focus on issues that would be obscured by more complex agent applications. Thus, it is useful to study them and their implementation techniques.

In this paper, I give a brief history of mail notification agents. I present experiences using Tcl as an agent-implementation language, describing advantages and disadvantages. I compare the results to similar agents. Finally, I discuss the use of Tcl as a configuration and interface language and describe some new techniques of interest to Tcl programmers.

BACKGROUND

Mail notification agents inform humans that electronic mail has arrived. There is a long history of such agents, including a large class of programs whose names include the syllable "biff" such as xbiff, xbiff++, pbiff, and of course, the original biff that appeared in an early BSD release of UNIX.

The basic idea of these programs is the same – to notify the user when mail has arrived. There are many variations and extensions of this idea. For example, some programs print a message on the console while others play an audio clip or perform an animation.

This draft is the last online version of the paper which appeared in *Software – Practice & Experience*, Vol 27(4), p. 481-493, April 1997. See the hardcopy version for the final paper.

Traditionally, biff-like programs were written as stand-alone tools. They were written in C and provided little flexibility in how they interacted with the user and other processes. And even contemporary biffs, while supporting polished visual or audio presentations, provide no means of communication with another program.

A HISTORY OF BIFF

In early versions of BSD, mail notification was accomplished by allowing mail delivery programs to write messages (e.g., "You have new mail") directly on the recipient's terminal.[4] Later, a dedicated daemon ("comsat") was created to encapsulate the common tasks such as figuring out to which terminal to write based on the username. The following snapshot shows sample output from comsat which prints out the important headers and the beginning of the body of the message.

```
New mail for libes@muffin has arrived:
----
From: rlb@us.teltech.com (Rodney Barnett)
To: libes@cme.nist.gov
Subject: Re: Automatic Login
Date: Thu, 20 Jul 95 10:07:37 CDT

Just a followup to let you know I got Expect working as you described.  I
was having trouble with a particular telnet client until I found what seems
...more...
```

Figure 1. comsat output

Simultaneous writing on a terminal by multiple programs invariably produced a corrupt display. Fortunately, this could be disabled by trivially changing the write permissions of the terminal device. There were a variety of ways to do this, but the biff program became a popular method. It is not clear why biff was written except perhaps that it was an excuse to use a rather distinctive and easily pronounceable name that stuck in people's minds. Among others, one legend has it that the author named biff after a colleague's dog and then enlisted the help of others to come up with a rationale for why the name made sense.[5,6] Whatever the reason, biff has entered common parlance for describing the class of programs that performs mail arrival and similar notifications.

Dilbert by Scott Adams

At the time of publication, United Media did not allow use of Dilbert in electronic form. Therefore, we have sadly omitted the humorous cartoon that appears in the final hardcopy paper. Readers can find it in on page 483 of the hardcopy of SP&E, Vol 27(4), April 97. (As an aside, the permission to reprint Dilbert in hardcopy cost us $60.)

Figure 2. another theory for the name "biff"

Modern biffs work quite differently than the original biff. Modern biffs periodically poll for changes, typically looking directly at the mail spool file. Polling places more of a load on the CPU than being signaled directly. However humans do not need instantaneous notification and so the polling cycle can be done infrequently with subsequent low cost. More importantly this design moves all of the control directly into the user's domain where no special permissions are required. This solution also solves the screen updating problem by using a dedicated display rather than an existing window.

While embodying characteristics that are generally accepted of agents, all of these prior implementations are limited in various ways.[7] Consider the following two biff implementations:

1. xbiff displays a mailbox icon with a flag that goes up when mail is present.[8] The C source is 112 lines but it is primarily a wrapper for the Athena mailbox widget.†

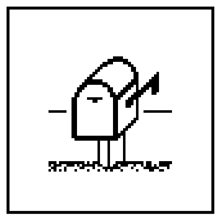

Figure 3. xbiff display

2. xbiff++ plays audio clips and displays bitmaps upon receipt of mail.[9]

Both of these programs are limited as to the displays and actions they can carry out. The most flexible, xbiff++, can play different audio clips and display different bitmaps upon reception of mail from different users. However, xbiff++ can take no other actions. For example, xbiff++ offers no way to display the messages nor is there any way to have it communicate its results with other programs. It is also highly nonportable since the routines are all "wired in". For example, the audio clips are only supported on Sun systems.

This nonportability and inflexibility is not surprising. Historically, biffs have been written in C, a compiled language that presents no special support in its programs for end-users to perform sophisticated control. Passing simple flags (e.g., getopts) or simple variable assignments (e.g., .Xdefaults) is common. It is difficult to justify designing (and writing and debugging) a specialized language specifically for such a simple tool. However appropriate for system applications, C is expensive in terms of programmer effort. As an example, the credits for xbiff++ list seven people with three singled out for the audio support alone, which only runs on a computer from a single vendor! The source code for xbiff++ is 3360 lines of C.

Because biff agents have historically not allowed their results to be communicated to other agents, many programs that are not biff agents provide their own biff-like behavior. For example, this is a common trait among news readers and serial-port communication programs. These programs are usually very complex in the sense that they do many related things – mail notification is just one more – and often present themselves as monolithic pieces of software. Monolithic software is good for users who have limited needs that the software meets. However, users with additional needs often find it difficult to extend or reuse such software because it was not intended for these purposes.

† Why does it take 112 lines of code for a single call? It doesn't. Most of the lines are used for comments, option handling, error handling, and of course, disclaimers and copyrights.

In contrast, programs such as shells and editors are usually quite configurable. However, because few people want to spend the time (or are capable of the programming involved), many shells (e.g., csh, zsh) and editors (e.g., emacs) have mail notification capabilities built in. Unfortunately, this built-in support is much like that of the monolithic programs – if it doesn't closely meet the needs of users, they are back to programming from scratch. Even worse, configuration of these large programs is hard because they are typically doing so many unrelated things simultaneously.

CONFIGURATION AND INTERFACE SOLUTIONS

This paper addresses two requirements that are important in notification agents. The first requirement is to detect situations and to carry out requests with enough flexibility that the user does not need to create yet another tool or to modify the notification agent. The second requirement is that the agent be able to interact not only with the user but with other agents or programs as well.

Tcl provides a solution to the configuration problem. Users can express their requests using Tcl. Unlike C, Tcl is interpreted and it is straightforward to allow end-users to express arbitrarily complex requests at run-time. A trivial request might be to display an image using Tcl statements[†]. Tcl is also capable of running other programs, interacting with other programs, using sockets, accessing files, etc.

Tcl also provides a solution to the interfacing problem. For example, a biff agent could communicate with an editor, asking it, for example, to display the contents of a message. By isolating notification in a single agent, other agents and the user environment in general is greatly simplified. Other random, seemingly unrelated tools do not have to know about the vagaries of mail formats, spool locations, or even what the user is interested in. That is all the domain of the mail notification agent. I return to this topic in more detail later in the paper (page 8).

TCL – OFTEN USED, BUT NOT TAKEN ADVANTAGE OF

It is relatively easy to build X window system clients using Tk. With the spread of Tk, it is only natural that several people have written Tk-based biffs. Surprisingly though, these have not taken advantage of the flexibility offered by Tcl upon which Tk is based. Instead these Tk-based biffs have been mere rewrites of the older C-based biffs. For instance:

3. tkpostage shows a picture of a U.S.A. metered stamp with a count of how many messages are present.[10] Another window contains a list of subject and author pairs. The source is 513 lines. The image data is another 6.5Kb. tkpostage is based on xpostage, which was written in 826 lines of C.[11]

4. tkpbiff performs an animation (a spinning postcard icon) and briefly pops up a window containing sender and subject.[12] The source is 506 lines. The image data is another 43Kb. tkpbiff is based on xpbiff, which was written in 632 lines of C.[13]

Again, all of these tools are limited in the actions they can take and the interactions they can have with other programs. Most importantly, these tools cannot affect any other programs and they can only assist the user directly.

† Displaying an image can actually be done in a single command. But the point is that any command or commands may be executed, not just an image display command.

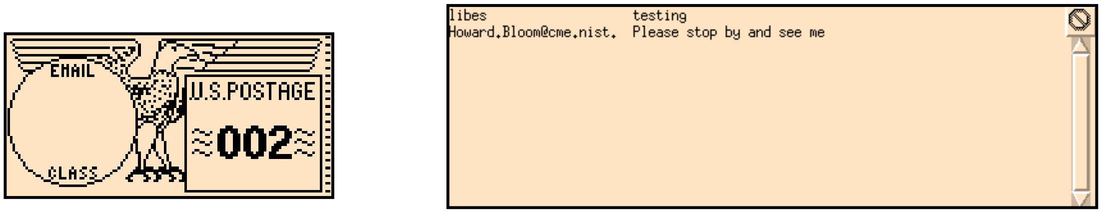

Figure 4. tkpostage displays

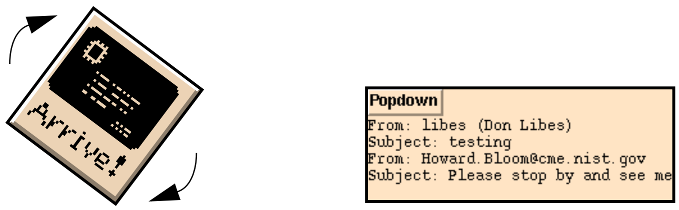

Figure 5. tkpbiff displays

In the remainder of this paper, I describe two agent programs that use Tcl effectively, allowing execution of arbitrary user actions upon relevant events. These actions may be communication with another program, creation of a new process, etc. Since any other program may be executed, it is possible to invoke native programs to perform tasks that would otherwise by nonportable. For example, playing an audio clip can be handled by a native program rather than routines built in to the agent.

TKNEWSBIFF – A NEWS NOTIFICATION AGENT

tknewsbiff is a Tcl/Tk-based agent for providing notification of new Usenet news. By default, tknewsbiff learns about user interests by examining a user-prepared file called .tknewsbiff. This configuration file contains initial declarations and attributes of newsgroups. For example, “watch” commands define newsgroups to be watched. For example:

```
watch dc.dining
watch nist.*
watch comp.unix.wizard  -threshold 3
watch *.sources.*       -threshold 20
```

For each newsgroup pattern, any newsgroup that matches it and which the user is subscribed to (according to their newsrc file) is eligible for reporting. By default, tknewsbiff reports on the newsgroup if there is at least one unread article. What is reported may be modified in various ways. For example, “-threshold 3” means there must be at least three articles unread before tknewsbiff will report the newsgroup.

The “ignore” command suppresses newsgroups that would otherwise be reported. For example, the following matches all comp.* and nist.* newsgroups except for nist.posix or .d (discussion) groups:

```
watch comp.*
watch nist.*
ignore nist.posix.*
ignore *.d
```

The flag “-new” describes a command to be executed when the newsgroup is first reported as having unread news. For example, the following lines invoke the UNIX “play” command to play a sound.

```
watch dc.dining -new "exec play /usr/local/sounds/yumyum.au"
watch rec.auto* -new "exec play /usr/local/sounds/vroom.au"
```

Both the “exec” command as well as the “watch” and “ignore” commands are Tcl commands. This provides great power in what can be done. For example, “-new” commands may be arbitrarily complex, perhaps calling user procedures or external commands. “watch” commands could be called from a loop or even from a -new event.

```
while {...} {
     watch $newsgroup -new "ignore $newsgroup"
}
```

The default action merely displays the newsgroup with the count of unread articles. An example is shown below.

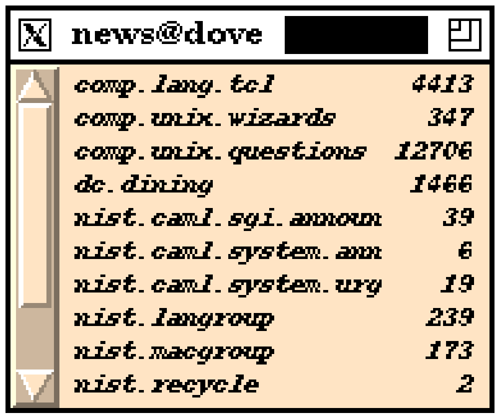

Figure 6. tknewsbiff display

INTERACTIONS WITH OTHER AGENTS

Built in to Tk is a high-level communication command called “send”. The “send” command supports application remote control of already executing programs (i.e, agents) that are also Tk-based. Commands may be sent including new procedure definitions (since process definitions are themselves commands). Security is possible to screen out inappropriate commands.[14] Command evaluation is handled as an event so that no explicit support is necessary for agents to offer send-based service.

As an example of inter-agent support, the “-new” action could direct a newsreader program to begin reading news in a different newsgroup. Or the agent could query the newsreader as to what newsgroup is currently being read so that it can be monitored more closely. For example:

```
watch $ng1 -new "send $newsreader newsgroup $ng2"
```

Similarly, the newsreader could in turn direct the news notification agent to monitor different newsgroups. For example, the newsreader could send watch commands for all newsgroups to which an article was crossposted. In this way, the user would be able to follow a web of related news automatically.

INTERACTIONS WITH THE USER

The initial rules used by the tknewsbiff are expressed as Tcl commands and stored in a configuration file. This contrasts sharply with most X tools, which use X resources.

No extra provisions are made for X resources beyond what Tk automatically provides. For example, it is possible to change the color of the widget backgrounds using the resource database, but non-Tk resources are not defined this way. Since users may invoke arbitrary Tcl/Tk commands, it is more general and powerful to phrase everything in terms of commands rather than through the X resource database.

tknewsbiff makes little use of command-line options since command-line options are very limited in power, comparable to that of X resources. About the only appropriate command-line option is one that directs the agent to its Tcl-based configuration file where much more complex configuration commands can appear.

Keystrokes (or mouse events) are interpreted by “bind” commands. For example, by default, button 3 (right) is bound to “unmapwindow”. The unmapwindow command causes tknewsbiff to remove the window from the display until the next time it finds unread news. (The mapwindow command causes tknewsbiff to restore the window.)

The user may continue to interact with the agent via the configuration file or the keyboard. The configuration file is reread periodically by the agent. Changing the configuration file is appropriate for complex commands that are to be permanent across future agents using the same configuration file. In contrast, temporary changes are typically made by interacting with tknewsbiff through pressing keystrokes and reading the display. Each of these are controllable to a large degree.

The display normally shows a scrollable list of pairs of newsgroup names and unread article counts. The basic idea of a list in a window is all that is automatically provided. This can be turned off entirely, or it can be modified in part. For example, it is possible to change what newsgroups are displayed. In particular, the watch command supports a -display flag, which describes a command to be executed every time the newsgroup is reported as having unread news. The command “display” is the default command. It schedules the newsgroup to be written to tknewsbiff’s display when it is next rewritten.

“display_list” is the list of newsgroups to be displayed at the next opportunity. This can be examined and modified by the user before the display is updated. While the display can be disabled totally or in part, it is built in to tknewsbiff program. I contrast this with the tkbiff agent described next in the paper.

There are a large number of other commands, options, and variables that tknewsbiff presents to the user. However, the details are not relevant to this paper.

One other noteworthy aspect is that tknewsbiff supports both common protocols for news access: 1) active file for directly mounted news file systems, and 2) NNTP for Internet news feeds. tknewsbiff uses Expect, an interactive automation tool, to provide access to NNTP, an interactive protocol.[15]

No directly comparable agents are known. A great deal of related agent work has occurred based around personal information filtering. The general approach is to scan the actual articles themselves looking for keywords, perhaps biased by recent newsreading history. The NEWT system is a good example of this.[16]

TKBIFF – A MAIL NOTIFICATION AGENT

tkbiff is a mail notification agent. Like tknewsbiff, tkbiff maintains an in-memory database of interesting information. In tkbiff, the information corresponds to the mail file and various information about it in an easily usable form. tkbiff tracks whether mail is created or deleted so that user-actions can be taken upon such events. This tracking is not trivial. For example, the user might concurrently use an MUA to change a message's status from "unread" to "read". Messages can also change location or order in a file.†

While tkbiff is similar to tknewsbiff as an agent, there are several interesting differences. The primary architectural difference between tknewsbiff and tkbiff is in how the GUI is provided. In tknewsbiff, the GUI is built in and the user is provided with numerous hooks with which to control the interface.

In contrast, tkbiff cleanly separates the GUI from the data collection responsibilities of the agent. Indeed, tkbiff has no GUI built in. The responsibility for providing a GUI, if any, is provided entirely by the user. tkbiff can actually be run with a pure Tcl interpreter since it has no knowledge of any particular GUI.

As with the tknewsbiff agent, tkbiff takes additional instructions from the user via a configuration file. Unlike tknewsbiff, any concepts of display lists, keyboard bindings, etc., are dealt with wholly in the configuration file. This simplicity permits the tkbiff program to be relatively short – about 300 lines. This can be smaller than a configuration file that provides a reasonably decent GUI.

Since building a GUI is actually a non-trivial task, a sample GUI configuration file is provided with tkbiff. In much the same style as tknewsbiff, a scrolling list of pairs – "from" and "subject" fields – are displayed. Example bindings provide popup displays of message bodies merely by pointing and clicking on a particular message.

If run using a Tk-enabled interpreter, tkbiff automatically installs the example GUI configuration file if the user does not already have one. If run using a pure Tcl interpreter, tkbiff uses a primitive interface, merely printing the "from" and "subject" fields to the standard output.‡

Both interfaces can be extended arbitrarily. For instance, a sound may be associated with the reception of a message similarly to the way it was accomplished in tknewsbiff. The following example demonstrates how the user might control a voice synthesizer ("speak") to announce messages.

```
proc announce_one_new_msg {from subject} {
     exec speak "$from sent you mail about $subject"
}
```

The announce_one_new_msg procedure is an procedure called and defined solely by the user configuration. For this reason, the user configuration can add additional arguments such as "cc" or "body" without changing the agent itself. Communication of data between the agent and the user configuration routines is made through a small set of functions and a database implemented via Tcl-based associative arrays.

TKBIFF INTERFACES

The user-supplied functions required by tkbiff are:

† An MUA ("mail user agent") is a program that manages activities involving sending and reading mail.

‡ "tkbiff" could more properly be called "tclbiff", however the "tk" prefix is what most people expect for Tcl-based software capable of supporting an X GUI.

announce_new_msgs – called when new messages arrive
renounce_msgs – called when messages disappear

tkbiff calls these functions as messages are added or deleted (by other agents). When called, the user functions browse through or update the database taking any action desired. The database is defined primarily by one array: msgs. For example, msg(1,from) contains the “from” field of the first message and msg(new_list) contains the indices of any new messages that have arrived. Using these interfaces, it is possible to emulate any other biff program. For example, a simple version of the original comsat-based biff is achieved with the following definition:

```
proc announce_new_msgs {} {
     global msg
     foreach id $msg(new_list) {
          puts "you have new mail from $msg($id,from)"
     }
}
```

Other actions are easily accomplished including adaptive behavior. As a simple example, an agent may observe that a user quickly read mail from some people or about some subject but not others. For these apparently interesting messages, the agent could vocalize more or all of the body, or could a start a mail reader on the user’s behalf. Wooldrige and Jennings mention even more complex mail notification tasks that are also solvable in the tkbiff framework.[7]

WELL-KNOWN MAIL PROBLEMS

tkbiff easily solves the mail recognition problems described by Arensburger and Rosenfeld.[17] They pose several scenarios in which a user (Bertie) wants automatic but different mail processing depending upon the source and content of the mail. Although A&R’s final goal is different, the mail recognition handling requirements are the same. Thus, I show A&R’s scenarios and how a user configures tkbiff to handle the scenarios.

Scenario 1: Was the mail addressed to Bertie personally or to a mailing list? (Any message that lists more than 10 recipients is considered to be a “de facto” mailing list message.)

The following two-part test solves this scenario. The first half of the test checks for explicit mention of “Bertie”. The second half checks whether the address has more than 10 recipients.

```
if {0 == [string match "*Bertie*" $to]
     || [llength [rfc822_split $to]] > 10} {
          ...
```

Scenario 2: Was Bertie the primary recipient of the message or did Bertie only appear in the “carbon copy” list?

The following two-part test checks for explicit mentions of Bertie in the To: and Cc: fields and offers different behavior for all possibilities:

```
if {[string match "*Bertie*" $to} {
          ...
     elseif {[string match "*Bertie*" $cc]} {
          ...
```

Scenario 3: If a message was addressed to Bertie directly, where did it come from? Any message that was sent from within Bertie’s institution is considered important and is brought to his atten-

tion. Bertie has also specified a list of "email friends" – friends, colleagues, VIPs – whose messages are considered to be important enough to be brought to his attention.

The following three-part test handles this scenario. The isfriend procedure is assumed to be a simple test that returns true if the sender matches one of the "email friends".

```
if {[string match "*Bertie*" $to] &&
        {[string match "*$institution*" $from]
    || [isfriend $from]}} {
        ...
```

CONFIGURATION MANAGEMENT IMPLEMENTATION TECHNIQUES

This section of the paper presents some experiences with Tcl as a configuration management language.

Tcl versus X Resources

1) The configuration language used by both notification agents is Tcl. This is exactly the language in which the agents are implemented. Thus, there is no need for an extra parser. The user communicates in the same language in which the agent "thinks".

Although Tcl is not a trivial language, the commands are straightforward for most tasks. For example, setting the width of a window could be as simple as:

```
set width 50
```

This could also be done using X resources, however X resources provide limited power. For example, suppose the user wants one resource to be twice the value of a second resource. The traditional solution requires both values to be updated separately because resource files do not provide any way of parameterizing values. There is no support for arithmetic, variables, procedures, etc. Since Tcl provides this, it is simpler to use Tcl for the entire configuration. Configuring the programs through X resources is still possible but is so much harder, there is no point.

One drawback of Tcl is that its power brings with it the demand for responsibility. Even users who are doing the simplest of configurations are in essence doing Tcl programming. Written in pure Tcl, neither tkbiff nor tknewsbiff have any significant mechanism for preventing users from wreaking havoc upon their internals such as by redefining procedures or data structures.

However, functions can be written to hide some of the detail that is unnecessary to the user's point of view. For example, the tknewsbiff agent maintains a list of newsgroups of interest to the user. Using list manipulation commands to add a new newsgroup requires the user to think about other things on the list and even the fact that they are maintained as a sequential list. In raw Tcl, this might look like this:

```
lappend watch_list comp.lang.tcl
```

In tknewsbiff, this operation was turned into a procedure call, which hides the irrelevant aspects:

```
watch comp.lang.tcl
```

A small procedure encapsulates this:

```
proc watch {args} {
    global watch_list
    lappend watch_list $args
}
```

Similar procedures encapsulate other user interfaces while still allowing the user full use of Tcl.

Providing an example configuration file automatically

As I mentioned earlier, the tkbiff agent includes an example configuration file. For many people, the default configuration will be sufficient and so the agent immediately installs the configuration file if no other is found.

To avoid dependencies on external files (that is, for the prototype configuration), the default configuration is stored within the agent. Much like Lisp, Tcl is effective at treating code as data which makes this particular problem easy to do. Alas, the obvious approach – storing the entire file as a list within the agent – was unsatisfactory because Tcl reformats backslash-newline sequences. There is no semantic difference after interpretation of these sequences. However, since the user is expected (or more precisely, encouraged) to modify the configuration file, it is helpful to preserve the original formatting of the file.

To avoid this pitfall, the agent opens its own source as a simple file, discards the source to itself, and reads the Tcl configuration commands at the end of the file. This approach seems inelegant, yet it is reliable and very fast.

This approach allows the program to be distributed as a single file. No installation and hence no Makefile is required. The agent contains help functionality and therefore requires no external documentation.

CONCLUDING NOTES

Agents should be able to gain and produce leverage from the strengths of other programs and agents. In order to achieve this, it is important to provide agents with sufficiently rich communication ability. This is provided naturally and easily by Tk. Agents should also have flexibility both in how users and how agents may describe what they expect from agents, and in how agents may adapt to changing situations. Tcl provides a framework that makes flexibility possible with a minimum of effort on the part of users.

Once communication and flexibility are provided, it is possible to modularize agents so that they can focus on their responsibilities without worrying about irrelevant issues. As agents and systems of agents increase in size, correct structuring of agents and their responsibilities will become more and more important.

The tknewsbiff and tkbiff programs demonstrate one way of accomplishing the dual goals of rich and easy communication and flexibility. We anticipate that our experiences building tkbiff and tknewsbiff using Tcl, Tk, and Expect, will prove helpful as we build new agents.

ACKNOWLEDGMENTS

Thanks to Neil Christopher and Peter Deno for discussion and assistance on this paper. Thanks to Dave Coombs, Ken Manheimer, Scott Paisley, K.C. Morris, and Sandy Ressler for valuable suggestions on the design of tkbiff. Thanks to Neil Christopher, Steve Osella, Shaw Feng and Steve Ray for many insightful discussions about agents. And thanks to Dave Fisher with the ATP Program for Component-based Software who provided the funding for the work described in this paper.

Thanks to Scott Adams for Dilbert. The strip is reprinted here with explicit permission of United Feature Syndicate, Inc.

AVAILABILITY

tknewsbiff is available and comes as an example in the Expect distribution. Expect and tkbiff are available via the web at:

`http://expect.nist.gov`

REFERENCES

1. Yezdi Lashkari et al, *Collaborative Interface Agents*, MIT Media Laboratory, 1995.

2. Michael Genesereth and Steven Ketchpel, Software Agents, *Communications of the ACM*, Vol. 37, No. 7, July 1994.

3. J.K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, ISBN 0-201-63337-X, April 1994.

4. William Joy, *Fourth Berkeley Distribution*, University of California, Berkeley, California, 1981.

5. Don Libes and Sandy Ressler, *Life With UNIX*, Prentice Hall, New Jersey, 1989.

6. Scott Adams, Project Biff, *Dilbert*, United Feature Syndicate, Inc., New York, New York, May 28, 1995.

7. Michael Wooldridge and Nicholas Jennings, Intelligent Agents: Theory and Practice, submitted to *Knowledge Engineering Review*, October, 1994.

8. Jim Fulton, Ralph Swick, *xbiff*, MIT X Consortium, 1988.

9. Jim Fulton, Ralph Swick, Mike Wagner, Jamie Zawinski, Lucid, Henry Spencer, Jef Poskanzer, Greg Earle, *xbiff*++, MIT X Consortium, 1990.

10. Dan Wallach, tkpostage, URL:ftp://ftp.aud.alcatel.com/tcl/code/tkpostage-1.3.tar.gz., November 15, 1993.

11. Cliff Herod, *xpostage*, Convex Computer Corp, 1989.

12. Benjamin Lurie, tkpbiff, *URL:http://www.cis.ohio-state.edu/hypertext/faq/usenet/tcl-faq/part4/faq.html*, 1993.

13. Kazuhiko Shutoh, *xpbiff*, InSoft System Lab., Yamaha Corp., 1990.

14. Nathaniel Borenstein, EMail With A Mind of Its Own: The Safe-Tcl Language for Enabled Mail, submitted to *ULPAA '94*, Barcelona.

15. Don Libes, *Exploring Expect*, O'Reilly and Associates, Sebastopol, California, January 1995.

16. Beerud Sheth, *NEWT*, Ph.D. thesis, MIT Media Laboratory, 1995.

17. Andrew Arensburger and Azriel Rosenfeld, To Take Arms Against a Sea of Email, *Communications of the ACM*, p 108, Vol 38, No. 3, March 1995.