inside:

IN THIS ISSUE
by Peter H. Salus, Special Issue Editor

CONFERENCE REPORTS
from the 6th Annual Tcl/Tk Conference

ANNOUNCEMENTS & CALLS
Calls for Participation for the 3rd USENIX Windows NT Symposium and for the 2nd LISA NT Conference

features:

An Architectural Overview of the ACE Framework
by Douglas C. Schmidt

A Tcl Story
by Guy Bobenrieth

Agenda: A Reminder Tool
by Rui Miguel Dias Anastacio

Managing Open Source Software
by Mark Harrison

Using Python
by Mark Lutz
### Upcoming Events

#### NordU99 - 1st Nordic EurOpen/USENIX Conference

**When:** February 9-12/99  
**Where:** Stockholm, Sweden

#### 3rd Symposium on Operating Systems Design and Implementation

Co-sponsored by ACM SIGOPS and IEEE TCOS  
**When:** February 22-25/99  
**Where:** New Orleans, LA  
**Who:** Margo Seltzer & Paul Leach  

#### 1st Conference on Network Administration

Co-sponsored by SAGE  
**When:** April 7-9/99  
**Where:** Santa Clara, CA  
**Who:** David Williamson & Paul Ebersman

#### 1st USENIX Workshop on Intrusion Detection & Network Monitoring

**When:** April 11-12/99  
**Where:** Santa Clara, CA  
**Who:** Marcus J. Ranum

#### 5th Conference on Object-Oriented Technologies and Systems (COOTS)

**When:** May 3-7/99  
**Where:** San Diego, CA  
**Who:** Murthy V. Devarakonda

#### USENIX Annual Technical Conference

**When:** June 7-11/99  
**Where:** Monterey, CA  
**Who:** Avi Rubin, Program Chair  
Clem Cole & John Heidemann, IT Coordinators  
Jordan Hubbard, Freenix Track Chair

#### 3rd USENIX Windows NT Symposium

**When:** July 12-16/99  
**Where:** Seattle, WA  
**Who:** Werner Vogels & Stephen Walli

#### 2nd Large Installation System Administration of Windows NT Conference (LISA-NT)

Co-sponsored by SAGE  
**When:** July 12-16/99  
**Where:** Seattle, WA  
**Who:** Gerald Carter & Ralph Loura

#### Eighth USENIX Security Symposium

**When:** August 23-26, 1999  
**Where:** Washington, D.C.  
**Who:** Win Treese, Program Chair  
Avi Rubin, IT Coordinator

#### 2nd Conference on Domain-Specific Languages

Sponsored by USENIX in cooperation with ACM SIGPLAN and SIGSOFT  
**When:** October 3-6/99  
**Where:** Austin, TX  
**Who:** Thomas Ball

---

For a complete list of future USENIX events, access [http://www.usenix.org/events](http://www.usenix.org/events)
FEATURES

15 An Architectural Overview of the ACE Framework
by Douglas C. Schmidt

26 A Tcl Story
by Guy Bobenieth

28 Agenda: A Reminder Tool
by Rui Miguel Dias Anastacio

31 Managing Open Source Software
by Mark Harrison

36 Using Python
by Mark Lutz

ANNOUNCEMENTS AND CALLS

44 3rd USENIX Windows NT Symposium

46 2nd Large Installation System
    Administration of Windows NT Conference

;login:
;login: is the official magazine of the USENIX Association.

;login: (ISSN 1044-6397) is published bimonthly by the USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710
$40 of each member’s annual dues is for an annual subscription to ;login:. Subscriptions for nonmembers are $50 per year.

Periodicals postage paid at Berkeley, CA and additional offices.

POSTMASTER: Send address changes to ;login:, USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

Editorial Staff

Special Issue Editor:
Peter H. Salus <peter@pedant.com>

Editor:
Rob Kolstad <kolstad@usenix.org>

Managing Editor:
Jane-Ellen Long <jel@usenix.org>

Copy Editor:
Eileen Cohen

Proofreader:
Kay Keppler

Designer:
Vinje Design

Typesetter:
Festina Lente

Advertising

Linda Barnett <barnett@usenix.org>

Membership and Publications

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710
Phone: 510 528 8649
FAX: 510 548 5738
Email: <office@usenix.org>
WWW: <http://www.usenix.org>

©1999 USENIX Association. USENIX is a registered trademark of the USENIX Association. Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this publication, and USENIX is aware of a trademark claim, the designations have been printed in caps or initial caps.
in this issue . . .

by Peter H. Salus
Special Issue Editor

For me, and most likely many others, Kernighan and Plauger's *Software Tools* (1976) opened up the world of "programs that make good tools." For others, like Debbie Scherrer and her group at Lawrence Berkeley Laboratories, it meant starting a "Software Tools User Group" as well. But these weren't the first "tools." IBM's JCL -- Job Control Language for the 360, still in use for the 390 -- may have been the first, as a scripting language; but all the UNIX shells are "little languages," used both for interactive commands and for scripts to automate frequent chores. REXX, PERL, and TCL fall into this group of scripting languages, too. And lex, yacc, grep, and troff are language tools.

Jon Bentley's brilliant "Little Languages" (CACM 1986; reprinted in vol. III of the *Handbook of Programming Languages*, 1998) had a tremendous impact, driving the creation of a number of more specialized languages (Peter Langston's talk at the 1986 Atlanta meeting of USENIX is a fine example). And even today we have USENIX conferences on "Domain-Specific Languages."

Nearly all of this grows out of one of the basic philosophical tenets of UNIX: Write programs that do one thing and do it well. If you're doing one thing, you don't have to generalize, and if you're focussed, the program will be a better one.

Furthermore, few little languages or tools are written by teams or committees; they are created by individuals who need something. (Brian Kernighan once said that AWK was the toughest thing he'd ever worked on, because three of them were working on it [Aho, Weinberger, Kernighan].)
I feel very strongly about neat tools and little languages, not merely because V6 UNIX was under 10K lines of code (Windows98 is said to contain 45M!), but because programming is (or should be) an art.

I have tried to find interesting materials to feature in this issue of jlogin, materials that are not rehashes of things previously written about in the bimonthly pages. I have also gone outside North America in an attempt to broaden our ambit.

The result of this, in part, is that I have not included any work on Perl nor Java: plenty has been printed in jlogin, and more than is needed elsewhere, about these.

I have begun the issue with an essay on ACE by Doug Schmidt, as I consider Doug's contribution to frameworks and design patterns of real importance. ACE is an object-oriented framework that provides core concurrency and distribution patterns.

Following this are two essays on Tcl, one about an attempt to get heterogeneous systems to cooperate, the other describing a handy little tool. I am very pleased that Guy Bobenreith (from Luxembourg) and Rui Miguel Dias Anastacio (from Portugal) have contributed work showing what's going on in the European Union.

Next, there is Mark Harrison's important "transitional" piece, "Managing Open Source Software." (As Mark refers to Perl at the beginning and the end of this essay, it may be the counterexample to my earlier statement.) But basically Mark, too, is writing about Tcl. (At this point, I should note that although I am Director pro tem of the Tcl/Tk Consortium, I have never actually met the European authors and my work for the Consortium has been salaried.)

Harrison mentions Python at the end of his article, so I asked Mark Lutz if I could reprint a slightly revised version of his contribution to the Handbook of Programming Languages. I am grateful to Mark for his revisions and to Macmillan Technical Publishing for letting me use the work. I think Python is a really interesting object-oriented scripting language.

It must be obvious that I am very interested in scripting and in languages in general. I hope that this special issue will be of interest to you.
6th Annual Tcl/Tk Conference

SAN DIEGO, CALIFORNIA
September 14-18, 1998

Overview
by Clif Flynt

This was Tcl/Tk’s “coming of age” conference. While previous years’ papers reported preliminary results and showed working prototypes, this year’s described systems that are now in production. The presenters of previous years’ papers tended to be interested in Tcl as a neat little language. This year, the presenters looked at several alternatives and chose Tcl as their development vehicle because it would let them devote their energies to solving a particular problem rather than fight with the language; in retrospect, they still believe that Tcl was the best choice.

The quality of the papers was very high. According to one of the members of the program committee, this was the first year that they had to reject good papers just because there was no time to present them.

Interaction with the attendees is always one of the reasons for attending this conference, and this year was no exception. Without a doubt this is the friendliest of the professional conventions I attend.

The Paradise Point Hotel was an excellent hotel, with a marvelous staff, though the swimming pools, saunas, and golf courses were somewhat wasted on our crew of techie geeks who were listening to papers all day and attending BOFs all night. The one thing the hotel lacked was a focal point. The rooms are spread around in little beach houses, and there is no central lobby/bar area where attendees can congregate. Since the Resort is not in a downtown area, we were limited to the one bar that was open late for socializing.

The night the bar had a live band, I just gave up and went to bed.

My personal high points of the convention (aside from the papers and general intellectual stimulation) were meeting the folks I’ve been corresponding and working with for the past year face-to-face, and meeting folks who learned Tcl with my TclTutor package. It’s a thrill to see folks that I virtually taught the language to attending the conferences. I’m really looking forward to the day when someone who learned Tcl with TclTutor delivers a paper at the conference.

KEYNOTE ADDRESS

Tcl/Tk, Agents, and Makin’ Pictures: A Whirlwind Tour

Michael B. Johnson, Pixar

Summary by Max Stevens

In a very entertaining talk, Michael Johnson told how Tcl has played a role in projects he has been involved in over the last few years. He began with his time as a student in the Media Lab at MIT and moved right through his years at Thinking Machines to his current work as a media arts technologist at Pixar, explaining in each case how Tcl has been a benefit.

Johnson began using Tcl back when it was almost still a gleam in Dr. Ousterhout’s eye. After reading a first draft of the first paper on Tcl, Johnson picked up Tcl 3.0 and started using it. The first problem he encountered was very serious but in fact had little to do with Tcl itself: It wasn’t Lisp, which is a big problem when you’re working in the Lisp-dominated Media Lab at MIT. But he needed an embeddable language that would work, so he persevered with Tcl and gained the status of weird machine guy, since he was able to get his projects working on new hardware quicker than anybody else, thanks to Tcl.
Session: Applications
Summaries by Clif Flynt

NBC's GEnesis Broadcast Automation System: From Prototype to Production
Stephen J. Angelovich, NBC Broadcast and Network Operations; Kevin B. Kenny and Brion D. Sarachan, GE Corporate Research & Development

Kevin Kenny opened this talk with an NBC promotional video describing the new all-digital NBC Broadcasting Operations Center. Kenny then described the "proprietary user-friendly software" (mentioned in the video) that makes it all work.

This was the third paper delivered to the Tcl/Tk community as this project has progressed. The first paper described how Tcl/Tk was being used to prototype the user interface as people were trying to figure out what they could do with the new digital technology. The second paper described the continuing development as the group created prototypes with more and more functionality and gathered more and more feedback.

This year, the system is in place at NBC, handling almost all of the daily broadcasts to the U.S., and people are still learning what can be done and asking for new features. Fortunately, the design of the system and the facility with which new Tcl functionality can be created make the continuing evolution of the system relatively painless. The original plan had been to develop the prototype with Tcl/Tk and then build the real system from more traditional components. However, like other presenters, Kenny and his group found that Tcl "just worked," so they never needed to recode the system.

Kenny also discussed several of the problems his group ran into while developing a package designed for 7x24 operation with a language that was really designed for small scripts. These ranged from performance issues to memory leaks. The performance issues were easily solved with the traditional Tcl approach of recoding the compute-heavy functions in C (or in this case, C++) and creating a new Tcl command to access the fast code. The memory leaks were a bit trickier. For example, they found that the underlying data structures that are allocated when a tag is created in a text widget do not get deleted when the tag is no longer in use. To solve this memory leak, they created a pool of tag names and used names from that pool, rather than generate unique tag names.

Kenny concluded his talk by pointing out that the flexibility of Tcl/Tk was what made the project work.

An Extensible Remote Graphical Interface for an ATM Network Simulator
Michael D. Santos, P. M. Melliar-Smith, and L. E. Moser, University of California, Santa Barbara

Michael Santos discussed the Tcl/Tk-based display that he constructed for a network simulator. He briefly described the Thunder & Lightning 40 Gbit/sec ATM network. His project is defining protocols to best utilize this high-speed network. At this network's operating speeds, the time to send a signal across the country at light-speed represents a large chunk of network capacity. For example, reserving space for a message by generating a request for space and then waiting for an acknowledgment could take longer than simply transmitting the entire message, simply because of the time it takes for a signal to go from San Diego to Boston.

Like other presenters, Santos is interested in Tcl not simply for the sake of the language; he uses Tcl because it lets him concentrate on his actual problem - getting data onto the wire - instead of the language he's using to implement the solution.
He pointed out that machine resources on a network simulator are limited and should all be dedicated to the simulation. If you can avoid running X Windows and a GUI-based monitor, you have more resources for your simulation. But GUI-based monitors with charts and graphs are very useful tools for monitoring a system’s behavior. He designed his system so that a small monitor lives on the simulator, and the user interface lives on a remote system and talks to the simulator via sockets. As a happy by-product of this design, the GUI monitor can also be used with the actual network switches.

One of the wins he got from Tcl and Tk was the extensions that would do what he needed. For example, by using BLT he got a chart recorder working much more quickly than would have been possible otherwise.

**WinACIF: A Telecom IC Support Tool Using Tcl/Tk**

David Karoly, Todd Copeland, and David Gardner, Advanced Micro Devices

David Karoly described how Tcl is used at AMD to support the SLAC (Subscriber Line Audio-Processing Circuit) family of programmable codec/filter IC chips that are used to build telephone linecards that interface between analog telephone and digitally switched networks. These chips perform A/D and D/A conversions, filter, compress, and expand the analog signals.

The chips can be programmed to control line power feed, ring, signalling and test functions. This gives hardware designers a great deal of flexibility, but with that comes complexity.

Tcl and Tk let the group of hardware engineers who best understood the chips design and construct a package for design engineers to use to configure and bench-test their boards. This package consists of three Tcl extensions to interface with the SLAC chips and the board being tested, and a data-driven GUI that configures itself to match the SLAC device being analyzed.

The users of this package are the silicon design team and AMD’s customers. WinACIF provides them with a GUI for interacting with all the programmable features of the SLAC devices. However, running all the tests they need to do on the latest SLAC devices requires a command language. From WinACIF, the users can create and execute Tcl scripts to automate these tasks.

When Tcl was extended with the libraries that program the SLAC devices, it provided the ideal solution. There was no need to develop a command language when Tcl provides a complete interpreter. Tcl is acceptable to the users because of its resemblance to other shell languages.

The WinACIF developers chose Tcl/Tk because it was a development platform that let them devote their energies to the problem of programming SLAC chips, rather than dealing with the complexities of developing MS-Windows applications.

**Charity Telethon Supported by Tcl/Tk**

Dave Griffin, Compaq Computer Corporation

Dave Griffin works with his local high school to help run their annual charity telethon auction. Last year, they automated this process using Tcl and Tk, and he described how this worked. Two-word summary: quite well. Griffin’s talk was aimed at the Tcl novices in the audience, showing how Tcl can be used “straight-out-of-the-box” to get from a problem to a solution in short order.

This high school, believe it or not, has two television studios, along with a network of PCs. The need was to coordinate the two studios, people taking bids over the phone, and Web interactions, and then to maintain the database of items, winners, and their bids, so that items could be delivered to the right recipients after the auction was complete.

The application was designed as a message-passing distributed system, with a master script that received and delivered messages, and various child scripts living on PC-class machines that sent their requests to the message-passing master script. The individual scripts were all small and simple. Griffin and his associate built a teleprompter using a simple wish script and large fonts, wrote a small memory-based database server (with journaling), and a message-passing controlling script. The database server, for example, required about 850 lines of code and generated reports in HTML to be read and printed with a browser. The web interface was created using the Tcl Web server.

Griffin provided several tips to script writers, including: Design for dynamic reload – scripts should check to see if a file is already open, etc., before attempting to open a file or create a window. This allows you to simply source a modified script and keep running. This is a great boon during debugging, and it lets you fix bugs on the fly. The Tcl namespace command makes it easy to create code modules.

**INVITED TALK**

**Tcl/Tk Update**

John Ousterhout, Scritps Corporation

Summary by Eric Melski

Ousterhout discussed the current state of Tcl/Tk and future routes for the language. The talk was mixed with the popular “Ouster-votes,” in which Ousterhout took rough polls from the audience on various topics.

He first examined what has happened with Tcl/Tk in the past year. At the top of the list was the move from SunScript to Scritps Corporation and the release of Tcl/Tk 8.0 in August 1997. Ousterhout also discussed the new Tcl/Tk 8.1 core, which, he explained, was on hold until
Scriptics could afford to develop it. New features include internationalization, thread safety, and improved regular expressions. An Oustervote revealed that about 10% of the attendees needed these new features.

The most interesting part of the talk was the statistical data on Tcl/Tk downloads, users, and applications. Over the past year, the number of Tcl/Tk downloads has roughly doubled. Most downloads were by Windows users; UNIX users were responsible for the next-largest portion; Macintosh downloads were the smallest of the three. Of the approximately 10,000 weekly downloads, 35% were by beginning users; 28% were by intermediate users; and 16% were by advanced users. 41% were for corporate use, and 24% were for hobby use.

Ousterhout noted that Tcl has become popular in "vertical" markets like dynamic Web content generation, finance, automated testing, and electronic design automation. A number of large companies use Tcl in these capacities, including AOL, CNet, Cisco, Motorola, and others.

Next, Ousterhout briefly described the Scriptics business model. He plans to balance open-source and commercial development. The core will remain free, but Scriptics will develop commercial Tcl/Tk development tools and provide Tcl/Tk support, training, and consultation. The first commercial product is TclPro, a step towards an IDE for Tcl. It features a debugger, syntax checker, and compiler. Future plans include adding a profiler, a GUI builder for Tk, and project-management tools.

Finally, Ousterhout discussed future plans for the Tcl/Tk core. He said that work was resuming on Tcl/Tk 8.1, and that he was already thinking about features for Tcl/Tk 8.2. A survey revealed that most users wanted drag-and-drop support and new widgets for Tk. Ousterhout said that many improvements to Tk were likely to come.

Ousterhout concluded that Tcl usage is continuing to grow rapidly. In an open forum attendees raised several issues, including the need to incorporate certain patches into the Tcl/Tk core, the need for printing support, and the need for better thread support.

**Session: Object Technology**

Summaries by Clif Flynt

**The Tycho Slate: Complex Drawing and Editing in Tcl/Tk**

H. John Reekie and Edward A. Lee, University of California, Berkeley

H. John Reekie described and demonstrated the Slate package, which extends the Tk canvas widget with objects better suited to implementing complex graphical editing and visualization than the standard primitive canvas objects. This package is written in pure Tcl (no C code) and allows a programmer to create complex canvas objects composed of several graphics primitives (mega-items). These mega-items support the concept of "shape" and may be associated with an "interactor." The "shape" facility allows an object's individual coordinates to be queried and modified (deforming the shape of an object) easily. The "interactors" provide a mechanism for linking higher-level event streams to an object. For example, the "follower" interactor follows the mouse cursor. This provides a simple mechanism for developing complex user interaction, such as graphical selection and drag-and-drop actions.

The Slate toolkit includes several new primitives such as 3-D boxes with labels and SmartLines. SmartLines are lines that know how to draw themselves from one object to another and can keep a label in an appropriate spot when the objects are moved and the lines need to redraw themselves.

This package looks like a very useful toolkit for developing graphical programming interfaces, flowcharting tools, network diagrams, etc. It's available at <http://ptolemy.eecs.berkeley.edu/~john/code/slate/>.

**Iclient/Iserver: Distributed Objects using [incr Tcl]**

Lee F. Bernhard, Bell Labs Innovations for Lucent Technologies

Lee F. Bernhard described a framework for sharing [incr Tcl] objects between remote processes in a client-server application. The Iclient/Iserver package allows a server written in [incr Tcl] to export [incr Tcl] objects. Clients written in [incr Tcl] can import these objects into their process and manipulate them as if they were local. In actual fact, the object continues to reside in the server process, and a stub is built in the client. The appearance of the stub, however, is identical to the parent, allowing the programmer to develop a client as a standalone application, and then easily convert it to a client.

This framework falls between the full-featured (complex) CORBA standard, and the low-level RPC primitives supported by Tcl-DP and the Tcl socket command. The Iclient/Iserver package handles low-level protocol problems like concurrency, locking, and watching variables (to trigger an event when a variable is modified by another task) in a way that is transparent to the application writer.

**Data Objects**

George A. Howlett, Bell Labs Innovations for Lucent Technologies

George Howlett presented a mechanism for allowing Tcl applications to construct high-speed special-purpose data objects that are more complex than the primitive Tcl data constructs, but with a more efficient implementation than writing a complex data object in pure Tcl.

Howlett described the performance problems with the BLT graph widget using lists to hold the graph vertices. These difficulties led him to add vectors to the BLT package. The techniques he used to create the vector array and make it act...
like the traditional Tcl data constructs provide a good example of how data objects should be separated from display and calculation code. The vector array has a high-speed (native) interface to interact with the C graph code, and a Tcl API interface to expose to the scripts.

Separating the data interface from the display and calculation code makes both sets of code simpler, allows simpler interfaces to be written for the data and calculation APIs, and allows the data constructs to be used by other display/calculation modules. Written out in simple English, this is a simple concept. But it is a design consideration that is so often overlooked when you’re designing a package and thinking of the package as a unit, instead of a collection of parts, that it bears repeating early and often.

Session: Testing and Debugging
Summaries by Clif Flynt

The extensible nature of Tcl and the ease with which Tcl applications can be modified make Tcl a good vehicle for developing test applications. The extensible nature of Tcl lets you add your custom driver code to the interpreter easily, while the available extensions provide analysis features. The number of papers in this session, and the number of testing- and simulation-related papers in other sessions, indicates that Tcl is being widely used in this area.

A Tcl-based Multithreaded Test Harness
Paul Amaranth, Aurora Group, Inc.

Paul Amaranth described a package developed by Aurora Group to perform load and regression testing on the Merit AAA Server, an authentication server based on the RADIUS protocol. Merit had a few low-level validation tools for use during development, but did not have a framework for testing the product under load or for long-term regression testing. Problems that needed to be addressed included: (1) The engineers are interested in developing RADIUS servers, not learning test tools and writing test scripts; (2) Test environments always need to be modified as new problems are discovered and new tools are developed. Tcl was chosen because it is so easily extensible and can be used to link the existing test packages. However, writing tests in pure Tcl violates requirement (1), and writing a test language violates requirement (2), since test languages tend to be rather rigid and nonextensible. The solution was to create templates that provide the setup and interfacing code. The engineer need only merge in required code to perform a given test. The 15,000 lines of RADIUS interface code were merged into the Tcl kernel as an extension. The multithreaded executive and template parser were written in pure Tcl code. Tcl is not the obvious language for these types of applications, but it has worked well, and the package is now being used in production.

Using Tcl/Tk for an Automatic Test Engine
C. Allen Flick, DSC Communications Corporation; James S. Dixon, Silicon Valley Networks Corporation

C. Allen Flick described a system developed at DSC to perform automatic testing via RS-232 and GPIB connections to the System Under Test (SUT). The immediate need was to upgrade an inhouse-written package that had been developed 10 years previously. Tcl and Expect allowed them to exceed the capabilities of that package with about two weeks’ effort.

Tcl supports building procedures on the fly, and this feature provided the group with a mechanism for creating test objects as necessary. The test object provides a framework that a nonprogrammer can use to construct a test for the DSC products. In order to support both batch and attended operation, the test framework is designed to run with only stdin/stdout, and a GUI is wrapped around the package to run it within an attended mode. The extensibility of Tcl allowed them to add hardware-specific interface code easily.

Their implementation allows tests to be written in a modular fashion and evaluated within the “TestExpert” test management system from Silicon Valley Networks. Tcl’s support for running scripts on multiple platforms means that the tests used in engineering can be used by the manufacturing group, and finally by field engineers.

wshdbg – A Debugger for CGI Applications

Andrey Vckovski, Netcetera AG

Andrey Vckovski described some of the difficulties in debugging CGI applications and how the wshdbg debugger can be used to debug Tcl-based CGI scripts such as those running within the wshb framework or written with NeoWebScript or Don Libes’s cgi.tcl.

The many mechanisms for generating dynamic Web content include CGI scripts, ISAPI and NSAPI modules for the Apache Web server, and Java servelets. CGI scripts are the original mechanism for generating dynamic Web pages. CGI scripts have some advantages, including stability, since each time a CGI script runs it runs as a separate process; the languages are vendor- and platform-independent; and CGI scripts behave the same when run interactively as when they are invoked from a Web server. Disadvantages of CGI scripts include the overhead of the fork/exec call, controlling the number of open IP ports, security concerns, and, of course, debugging the scripts.

The wshdbg debugger is a Tk-based remote debugging package with windows to display the environment variables, command-line parameters, and information about the current state of the script being debugged. The script being
debugged must include a stub with the remote interface code, but otherwise behaves in a normal manner. The debugger allows you to set breakpoints and variable traces, at which point the debugger takes control of the script, and you can examine the state of the CGI script, evaluate Tcl commands in the CGI script environment, and view a log of events.

People familiar with xgdb and dbxtool will find this a rather stripped-down debugger. People who have been struggling with "printf" debugging for their CGI scripts will appreciate having a debugger that gives them the basic interactive debugging tools.

One of the nice quotes from this talk was: "Software engineering in the context of Web applications is sometimes still far away from state of the art." This package starts to move CGI script debugging out of the distant past.

Session: Web Technology (Server-Side)
Summaries by Clif Flynt

NeoWebScript: Enabling Web Pages with Active Content Using Tcl
Karl Lehnenbauer, NeoSoft, Inc.

Karl Lehnenbauer has been active in the Tcl community for almost forever. In this talk he described how the NeoWebScript package makes creating active Web content fast and easy.

Since Apache is designed to be extended and Tcl is designed to be embedded, it was relatively easy for NeoWebScript to leverage the power of these platforms to provide a secure, efficient server-side scripting language. NeoWebScript supports normal server features like cookies and SSL, as well as features like parsing form data, sending mail from the server, posting news from the server, embedding rotating banner ads and counters, and accessing databases. NeoWebScript allows a programmer to create forms on the fly with a forms package that uses sets of key/value pairs, and can parse the filled-out form into an associative array using the keys as array indices.

NeoWebScript is in use at about 1,200 sites and is available from <http://www.neosoft.com/neowescript>.

TclXML: XML Support for Tcl
Steve Ball, Zveno Pty Ltd

Steve Ball described his work creating an XML parser for the Tcl language. XML is gaining support as a text-description language. It can be considered (in a rough sense) as the good parts of HTML and SGML. An XML document can embed both display instructions (as HTML does) and data descriptions. With the wealth of data that can be in an XML document, there are multiple methods that one may wish to use to view the document. You may want to ignore the data descriptions and only use the display instructions (for a browser-type application), or ignore the display instructions and only evaluate the data descriptions (for instance, if you wished to make a database of all chapter headings).

TclXML uses James Clark's nonvalidating XML parser to add XML parsing to the Tcl language. Ball's previous projects (Plume, the Tcl Web Browser) parsed a document into a nested list. For the XML project he's abandoned that approach in favor of a Tcl-handle-based architecture. The problem with nested lists was that it is difficult to walk up and down a list, and the DOM (Document Object Model) requires that a list be dynamic, not static, as worked for his previous parsers.


Creating High Performance Web Applications using Tcl, Display Templates, XML, and Database Content
Alex Shah and Tony Darugar, Binary Evolution, Inc.

Tony Darugar described Binary Evolution's Velocigen package for generating dynamic Web pages. The Velocigen package solves the problem of fork/exec overhead in CGI scripts with a client-server model. The Velocigen engine runs separately from the Web server to service CGI requests. Velocigen uses Tcl to link the database engine and Steve Ball's TclXML, allowing the system to extract an XML document from the database, parse the XML document into a tree, and finally map the XML-tagged information to HTML tags. One of the advantages of this technique is that by divorcing the content from the presentation you can modify the display without modifying the base data simply by changing the XML-to-HTML mapping functions.

Darugar demonstrated a technique that can be used to describe content with XML using a carrot cake recipe as the example. In this example, the ingredients are tagged as <ingredient>, making it easy for an automated engine to parse the page for content and index the content in a database. When the page is displayed, the <ingredient> tags are mapped to HTML tags.

Generating HTML pages on the fly with Velocigen is not as fast as having pre-generated HTML pages, but is much faster than generating pages with CGI scripts. More information on this package is available at <http://www.BinaryEvolution.com/>.
Session: Web Technology (Client-Side)

Summaries by Clif Flynt

The Tcl plug-in for Netscape was released a couple of years ago. There has been a need for more development in this area, and these papers show that people have been working to extend Tcl's capabilities in this area.

WebWiseTclTk: A Safe-Tcl/Tk-based Toolkit Enhanced for the World Wide Web

Hemang Lavana and Franc Brglez, North Carolina State University

Hemang Lavana described the WebWiseTcl and WebWiseTk packages that extend the Tcl/Tk plug-in to be more useful for Web programming applications.

The Tcl/Tk Netscape plug-in released by Sun Research Labs had some restrictions that were due to security concerns. The most limiting was that top-level windows and menus were not supported because (1) new top-level windows can fill up a user's screen; (2) a menu can be torn off to become a new top-level window; and (3) a menu does a global grab of events while it is displayed. Allowing the creation of new windows and allowing an application to grab all events provides mechanisms for denial-of-service attacks.

WebWiseTk solves the new-window problem by rerouting requests to create a new top-level window into a procedure that simulates creating a top-level window by creating a set of frames. This restricts the new top-level windows to the area of the original browser window and removes the possibility of filling a screen with new windows. The problem of the global grab is solved by restricting the grab function to local grabs. Grabbing only events destined for a Tclet does not provide a mechanism for disrupting any other tasks.

WebWiseTcl allows Tcl scripts to be written into several smaller, manageable script files, by modifying the auto_load mechanism to "source" files from a URL (Tcl's host site). A few commands, such as exec, load, and send, cannot be implemented without compromising security of the plug-in. The goal of WebWiseTclTk is to support the complete remainder of the Tcl/Tk command set while maintaining security. They have achieved this goal. Using WebWiseTclTk, developers can now also "wrap" existing multi-window Tcl-based applications for immediate Web access as Tcllets — providing full capabilities of the original application but without rewriting the existing Tcl code.

More information on this package is available at <http://www.cbl.ncsu.edu/software/WebWiseTclTk>.

Internet-based Desktops in Tcl/Tk: Collaborative and Recordable

Amit Khetawat, Hemang Lavana, and Franc Brglez, North Carolina State University

Amit Khetawat described and demonstrated the "ReubenDesktop," an application suite that allows several users on the Internet to share drawing and message windows, record the interaction, and later replay this session.

The collaborative desktop includes a primary drawing surface, which can only be accessed by a single user at a time, and message exchange boxes, which all users can use for broadcasting a message to all other users. Access to the drawing surfaces is controlled by a token, which can be passed from user to user. The holder of a token can modify the drawing surfaces, at which point the master copy of the surface is updated. The session master then updates all the other instances of the surface.

These updates are performed by sending Tcl scripts to the participating desktops. The Tcl commands to update a surface are also saved to allow a session to be replayed. All users are allowed access to the message exchange box so that users can communicate with one another simultaneously.

Khetawat showed an example of how the ReubenDesktop can be used to design a custom IC chip. In this situation, the final customer, the IC design staff, and the manufacturing site are frequently separated by thousands of miles. The ReubenDesktop can allow these users to work together in real-time, or to review the actions of others by replaying a session.

More details on this package are available at <http://www.cbl.ncsu.edu/software>.

Creating a Multimedia Extension for Tcl Using the Java Media Framework

Moses DeJong, Brian Bailey, and Joseph A. Konstan, University of Minnesota

Moses DeJong described techniques for merging Tcl and Java using the Java Media Framework as an example of what the Tcl community can gain from this merging of technologies. The Java Media Framework provides a robust multimedia engine, functionality the core Tcl interpreter lacks. Tcl users can leverage this support using TclBlend on Jacl.

The Tcl interface to the JMF allows a programmer to add multimedia support to a Tcl application with just a few lines of Tcl code. DeJong demonstrated a simple script which linked playing a sound to clicking a button.

Java extensions are more platform-independent and less prone to memory leakage than the traditional C-code extensions. However, the 1.0 Tcl/Java package from Sun has some deficiencies. The primary problem is that the object-type information is lost when a Java object is exported into the Tcl interpreter. This poses a problem for tracking inheritance and overloaded methods.

DeJong has developed code that overcomes these deficiencies. That code has now been merged into TclBlend and JACL.

Visualizing Personal Web Caches with Caubview
Charles L. Brooks, GTE Internetworking; Murray S. Mazer, Curl Corporation; Frederick J. Hirsch, The Open Group Research Institute

Charles Brooks described Caubview, a package for viewing the HTML page caches created by the CaubWeb system. The CaubWeb system is designed to provide disconnected Web access by pre-fetching and caching HTML pages. The CaubView package allows the developers and users to analyze how well CaubWeb is doing at predicting what pages will need to be fetched.

Tcl/Tk was chosen as the platform for this development partly because of the “Publish or Perish/Demo or Die” culture rampant in today’s research and development departments. This choice has been a mixed blessing. On one hand, Tcl/Tk has enabled the group to leverage other people’s work by using extensions to gain required functionality, instead of needing to write all the code from scratch. On the other hand, extensions tag the Tcl core, there is no library of critical analysis of the existing extensions, and sometimes an extension becomes orphaned when the original author moves on to other tasks.

Tcl/Tk met the need for a cross-platform development environment, and the code has ported easily across the UNIX and Windows platforms. But the Windows platform support is still growing and changing slightly, and ports between revisions of TCI/Tk chewed up more time than the group would have liked. Overall, using Tcl/Tk has allowed the group to develop and test different methods of presenting the cache information to users, rather than worrying about language issues, but Brooks feels that the Tcl environment could be better.


Work-in-Progress Sessions
Summaries by Max Stevens

Demonstration of TclPro: Development Tools for TCI
Ray Johnson, TclPro Engineering Manager

This was one of the most anticipated WIPs, since it included a demo of the debugger in the newly released TclPro product from Scriptics. It began with a short description of features for each of the products in the TclPro package, and moved into a well-received demo of the debugger on Windows NT.

tkTable v2
Jeffrey Hobbs, Siemens

tkTable is a complex mix of entry, text, and listbox widgets, and is a feature-packed powerful widget that works on all platforms that Tk does. Hobbs described some of the main features and limitations of the table, ending with a short discussion of some of the future directions for tkTable, including PostScript output.

Tclsh
Michael McLennan, Bell Labs Innovations for Lucent Technologies

The Tcl Install Shell originated from the need for an integrated install program for the Tcl Blast CD, but it was designed to be very generic and powerful. Features include a configurable list of packages, dependencies noted among those packages, and the ability to patch both ASCII and binary files, all packaged in a user-friendly GUI.

New Features in BLT
George Howlett, Bell Labs Innovations for Lucent Technologies

The new features in BLT include a powerful new hierarchy widget, as well as new features for images, such as arbitrary scaling. A tabbed notebook widget was also added with a feature to tear a tab page out of the widget into a new top-level window and to put it back in later on; this was very well received by the audience. For future work, he mentioned in passing that he was working on a solution to one of the more frustrating aspects of Tk – the lack of a printing widget.

Patching Tcl/Tk Activities
Leo Schubert, Brückner & Jarosch

Leo Schubert described some of the core patches used at Brückner & Jarosch. The most impressive of these include a patch for Tk 8.0 that speeds up most applications by a factor of 2 to 5, giving a big performance boost especially to canvas and BLT users. In addition, work was done on “winico,” an extension for Tk 8.0 on Windows that allows users to set the top-level icon in the upper left corner of the window. He hoped to get these changes into the core. For more info, see <http://ftp.bj-ig.de/pub/tcltk>.

Birds-of-a-Feather Sessions
Summaries by Max Stevens

[inrc Tcl] - Object Oriented Programming

The introduction of version 3.0 of [inrc Tcl] figured prominently in this BoF, led by the creator of [inrc Tcl], Michael McLennan. The main new feature that met with great approval is the ability to dynamically load [inrc Tcl] as an extension into Tcl 8.0.3. And, with [inrc Tcl] taking advantage of the byte code compiler and being distributed with the TclPro package from Scriptics, [inrc Tcl] may soon become a standard part of all...
new Tcl programs. In the second half of the talk, McLennan went through some of the porting issues from previous versions of [incr Tcl] to the new version 3.0. Most of these involved modifying scripts to use the new namespace facilities in the Tcl core, since the namespaces were implemented slightly differently from how they were in [incr Tcl].

The BoF ended with a few ideas for the next major release of [incr Tcl], and then broke up to allow McLennan to replace his ever-present yet now empty bottle of Mountain Dew.

For more information on [incr Tcl], see <http://www.tcltk.com/tcltk>. For information on porting issues to Tcl 3.0, see <http://www.tcltk.com/tcltk/tcl3-port.html>.

Scriptics BoF – Jacl and Tcl Blend

Led by Bryan Surles from Scriptics, this BoF began with a short explanation of the differences between Jacl and Tcl Blend, and led into the motivation behind the products and where they are headed in the future. Jacl, a Tcl interpreter written entirely in Java, is perhaps three-quarters finished and is designed to give the predominantly Java programmer a scripting interface. Tcl Blend takes a slightly different approach by wrapping Java APIs, giving the more traditional Tcl programmer access to Java and allowing for easy integration between C and Java programs. Both were designed with the idea that Tcl scripts could leverage the cross-platform nature of Java and take advantage of Java's growing popularity.

The discussion focused on the ability of these products to run as applets, and in doing so perhaps replace the functionality of the Tcl plug-in for Netscape. Neither Jacl nor Tcl Blend works as an applet yet, since the restrictions imposed by the Java security safeguards make this a tricky prospect. Regardless, many participants emphasized that this problem must be solved, since the lack of applet support was keeping them from switching to Jacl and Tcl Blend from the plug-in. With the release of TclPro 1.0, it is expected that Scriptics will be able to spend more time on Jacl and TclBlend in the coming months to solve this and other issues.

For more information, see <http://www.scriptics.com/jacl>.

INVITED TALK

Yacc Meets Tk?

Steve Johnson, Transmeta Corporation

Summary by Eric Melski

Steve Johnson discussed the possibility of using a Yacc-like tool to rapidly develop graphical user interfaces in much the same way that Yacc is used to create command-line interfaces.

Ten years ago, most programs used text interfaces. Each interface had its own syntax, and little prompting or user help; errors drew "Syntax Error" messages from the program. Yacc was a powerful tool for creating these interfaces, but they were hard to use. By three years ago, most programs had a GUI. Menus and icons were easier to use, choices were prompted, and there were no "Syntax Error" messages. GUIs were clearly an advance. Now, however, we are slipping. Menus and icons are becoming more complex, and many visible choices are in fact illegal. "Syntax Error" messages have been replaced by a loud "BEEP!" Modern interfaces have a more difficult task than earlier interfaces. They have to deal with different classes of users, from beginners to experts, and many interfaces have multiple screens and multi-level menus. Johnson proposed using a Yacc-like tool to create GUIs based on a grammar, allowing the easy design of properly functioning GUIs that can accommodate a wide range of users.

He made a convincing argument for the use of a grammar as the basis for the interface. There is a lot of theory about grammars, and the grammar can document the interface and enforce consistency. Unfortunately, traditional grammars are inadequate. Temporary events are difficult to model, as are state changes that affect the interface. Instead, Johnson suggested a token-based grammar, in which events such as key and button presses would be tokens. Widgets would produce events, and the grammar would allow legal events. The grammar would automatically take care of making only valid options visible.

A GUI grammar, therefore, would consist of five parts: TOKENS, symbols, <booleans>, {actions}, and rules. TOKENS correspond to events; symbols represent states or goals in the interaction; <booleans> are used to reflect state information such as the system state or the user experience level; {actions} are fragments of code that are executed when rules permit; and rules are similar to those in a traditional grammar.

His proposed tool, GUYACC, is now only an idea. However, its construction should be straightforward. Building rule sets is something with which many are familiar, and many of the ideas behind the new grammar are just extensions of the traditional grammar. Johnson is interested in taking on the challenge of writing GUYACC, but he wants help from others in the community.
Session: Language Issues
Summaries by Max Stevens

Using Content-Derived Names for Package Management in Tcl
Ethan L. Miller and Kennedy Akala, University of Maryland, Baltimore County; Jeffrey K. Hollingsworth, University of Maryland, College Park

Installation and distribution of programs are big problems. There is no easy way to perform these tasks, even though they are routinely done. Whether it’s installing a new version of a program over an old one, thereby removing the old version before the new one even works yet, or changing the third-party software packages that are required for the product to work properly, there is no known easy solution. Ethan Miller presented a method of solving these problems through the use of content-derived names for package management in Tcl.

The idea is simple. Instead of human-readable names, packages are given a digital signature (or a cryptographic hash value) for their name. This way the integrity of the package can be assured, since upon receipt of the package you can simply run the file through the same hash function and compare the result to the file name. If they match, the package has not been tampered with. Once you have the root package (which must be trusted), it will contain references to other packages it requires. These references will be in the form of digital signatures, so you know that if you get a package of the same name, it can be trusted as well. This allows you to distribute the root package of a program, and from that all of the dependent packages can be retrieved from the network automatically.

The process is recursive, so even the new packages retrieved automatically can have their dependent packages automatically retrieved until all necessary packages are in place. In addition, since digital signatures are guaranteed to be unique, new versions of products can be installed in the same directory as old versions without fear of name clashes.

In order to distribute Tcl software using this technique, you need to follow only a few rules. First, you must use namespaces for all of the packages you write. Second, a package must be encapsulated entirely in one file. Third, you must be careful to avoid mutually recursive packages; the dependency graph must be a directed acyclic graph. Finally, use external packages as you need them. When it comes time to distribute your code, simply run a postprocessor over all your files that will generate the digital signatures, replace the embedded package names with these signatures, and then rename all the files with the appropriate signatures.

Thereafter, distribute the root package and make sure all dependent packages are available.

Miller has this working for Tcl packages and is nearing completion on a system that works with Linux binaries. He hopes to extend these ideas to more languages in the future and to streamline the process of generating and distributing the files. For more information, see <http://www.csee.umbc.edu/~elm/Projects/CDN/>.

Using Tcl to Rapidly Develop a Scalable Engine for Processing Dynamic Application Logic
Greg Barish, Healthco Corporation

Greg Barish described an Internet-based Tcl application in use at Healthco Corporation that uses a rule-based system to manage health benefits for employers. The idea was to have a system that is oriented toward the individual, with rules executed on a per-individual basis (such as “if [\$age > 21] \$eligible = 1”).

The requirements for this system were quite demanding. First, it had to be very fast, since it was expected that the application would undergo periods of extremely high use by many employers at once; with each employer having to update benefits for several employees – each having potentially several rules to be evaluated – a speed lag was not acceptable. Second, since benefit management could quickly change (e.g., through the addition of several new rules at once), an extensible and easily scalable solution was required. Finally, an interface to a database was required in order to access the necessary information.

The speed issue was addressed by writing speed-critical extensions in C and adding them to the Tcl interpreter. As always, there was a tradeoff here between the specific commands written in C, which would be much harder to maintain but faster, and the more generic commands written in Tcl, which are easier to understand and maintain but not as fast. The extensibility and scalability problems were primarily dealt with through a careful design and architecture of the overall program. This included the use of concurrent execution of rule evaluators, called “motors,” in order to maximize the number of rules that can be evaluated at once. The database interface was already supplied through Tcl extensions such as OraTcl and SybTcl.

The end result was a rapidly developed rule engine that successfully met all criteria. The rapid development time allowed the company not only to quickly change the implementation to match changing design requirements, but also to reallocate resources quickly to other tasks such as system integration. The ease in extending Tcl was another benefit that aided in massaging the system toward the final design specifications. As for drawbacks of using Tcl, only one was noted, and it was expected: a slight hit in speed. However, the time to develop a unique system in a lower-level language such as C would have taken precious resources away from equally important tasks such as overall system integration.
Using Tcl to Script CORBA Interactions in a Distributed System

Michael L. Miller, Advanced Micro Devices; Srikkumar Kareti, Honeywell Technology Center

Michael Miller gave a brief overview of the design process for a system to be used at Advanced Micro Devices, and how Tcl fits into that design. The company needed a system to reduce time, cost, and integration effort to deploy an Advanced Process Control (APC) solution into its semiconductor manufacturing facilities (fabs). This system would have to fit into the current framework, which consisted of distributed object-oriented components using CORBA messaging services.

It was determined that a scripting language was needed because the people using the system were control engineers, not programmers, and they needed the flexibility and functionality to implement any number of APC applications. The criteria for the language were that it must be easy to use, easily embeddable (since it was going to be embedded in a CORBA server), able to deal with CORBA to a degree, easily extensible, and cross-platform. Tcl, in comparison to Perl, Python, and Visual Basic, won easily.

In the APC framework, the Plan Execution Manager (PEM) acts as the choreographer component, performing “APC” at runtime for a particular process. Tcl was used by having the PEM create a new Tcl interpreter to evaluate “sub-scripts” for each “APC recipe” whenever a batch of silicon wafers was ready to be processed. The Tcl scripts defined not only what the APC framework did, but also in what order. Extensions were written for Tcl since the available extensions were deemed to be too general (and therefore harder to use), and some weren’t available for all platforms. Here the extensibility of Tcl was a real bonus.

In general, the use of Tcl was seen as a great asset, and it meshed well with the CORBA environment. The only major concern Miller mentioned was the lack of thread support (this was overcome by using a crude mutex lock around the entire Tcl library), but he noted with satisfaction that thread safety is being addressed in Tcl 8.1.
an architectural overview of the ACE framework

A Case Study of Successful Cross-Platform Systems Software Reuse

Communication software for next-generation distributed applications should possess the following qualities:

- Flexibility is needed to support a growing range of multimedia datatypes, traffic patterns, and end-to-end quality of service (QoS) requirements.
- Efficiency is needed to provide low latency to delay-sensitive applications, high performance to bandwidth-intensive applications, and predictability to real-time applications.
- Reliability is needed to ensure that applications are robust, fault tolerant, and highly available.
- Portability is needed to reduce the effort required to support applications on heterogeneous OS platforms and compilers.

This article describes the software architecture of ACE [1], which is a freely available, open source C++ framework targeted for developers of high-performance and realtime communication services and applications.

The ACE framework provides an integrated set of components that help developers navigate between the "Scylla and Charybdis" limitations of (1) low-level native OS APIs, which are inflexible and nonportable and (2) higher-level distributed object computing middleware, which is often inefficient and unreliable. This article describes the structure and functionality of ACE, outlines several complex communication middleware applications that have been developed with ACE, and summarizes the key lessons learned developing and deploying the reusable object-oriented (OO) communication software components and frameworks in ACE.

Overview of ACE

ACE is an OO framework that implements core concurrency and distribution patterns [2] for communication software. ACE provides a rich set of reusable C++ wrappers and framework components that are targeted for developers of high-performance, realtime services and applications across a wide range of OS platforms, including Win32, most versions of UNIX, and many realtime operating systems. The components in ACE provide reusable implementations of the following common communication software tasks:

- connection establishment and service initialization [3]
- event demultiplexing and event handler dispatching [4-6]
- interprocess communication [7] and shared memory management
- static and dynamic configuration of communication services [8, 9]
- concurrency and synchronization [5, 10]
- distributed communication services – such as naming, event routing [2], logging, time synchronization, and network locking
- higher-level distributed computing middleware components – such as Object Request Brokers (ORBs) [11], Web servers [12], and electronic medical imaging systems [13].

The Structure and Functionality of ACE

The ACE framework contains ~150,000 lines of C++ code divided into ~450 classes. To separate concerns and to reduce the complexity of the framework, ACE is designed using a layered architecture. Figure 1 illustrates the relationships between the key components in ACE.

The lower layers of ACE contain an OS adapter and C++ wrappers that portably encapsulate core OS communication and concurrency services. The higher layers of ACE extend the C++ wrappers to provide reusable frameworks, self-contained distributed service components, and higher-level distributed computing middleware components. Together, these layers and components simplify the creation, composition, and
configuration of communication systems, without incurring significant performance overhead. The role of each layer is outlined below.

The ACE OS Adaptation Layer

The OS adaptation layer constitutes approximately 13% of ACE, i.e., \( \sim 20,000 \) lines of code. This layer resides directly atop the native OS APIs written in C. The OS adaptation layer shields the other layers and components in ACE from platform-specific dependencies associated with the following OS APIs:

- **Concurrency and synchronization.** ACE’s adaptation layer encapsulates OS APIs for multithreading, multiprocessing, and synchronization.

- **Interprocess communication (IPC) and shared memory.** ACE’s adaptation layer encapsulates OS APIs for local and remote IPC and shared memory.

- **Event demultiplexing mechanisms.** ACE’s adaptation layer encapsulates OS APIs for synchronous and asynchronous demultiplexing I/O-based, timer-based, signal-based, and synchronization-based events.

- **Explicit dynamic linking.** ACE’s adaptation layer encapsulates OS APIs for explicit dynamic linking, which allows application services to be configured at installation-time or run-time.

- **Filesystem mechanisms.** ACE’s adaptation layer encapsulates OS filesystem APIs for manipulating files and directories.

The portability of ACE’s OS adaptation layer enables it to run on a wide range of operating systems. The OS platforms supported by ACE include Win32 (WinNT 3.5.x and 4.x, Win95, and WinCE using MSVC++ and Borland C++), most versions of UNIX (SunOS 4.x and 5.x; SGI IRIX 5.x and 6.x; HP-UX 9.x, 10.x, and 11.x; DEC UNIX 3.x and 4.x; AIX 3.x and 4.x; DG/UX; Linux; SCO; UnixWare; NetBSD; and FreeBSD), real-time operating systems (VxWorks, Chorus, LynxOS, and pSOS), and MVS OpenEdition.

Because of the abstraction provided by ACE’s OS adaptation layer, a single source tree is used for all these platforms. This design greatly simplifies the portability and maintainability of ACE.

The ACE C++ Wrapper Layer

It is possible to program highly portable C++ applications directly atop ACE’s OS adaptation layer. However, most ACE developers use the C++ wrappers layer shown in Figure 1. The ACE C++ wrappers simplify application development by providing type-safe C++ interfaces that encapsulate and enhance the native OS concurrency, communication, memory management, event demultiplexing, dynamic linking, and filesystem APIs.

The C++ wrappers provided by ACE are quite comprehensive, constituting \( \sim 50\% \) of its source code. Applications can combine and compose these wrappers by selectively inheriting, aggregating, and/or instantiating the following components:

- **Concurrency and synchronization components.** ACE abstracts native OS multithreading and multiprocessing mechanisms like mutexes and semaphores to create higher-level OO concurrency abstractions like Active Objects [10] and Polymorphic Futures [14].

- **IPC and filesystem components.** The ACE C++ wrappers encapsulate local and/or remote IPC mechanisms [7] such as sockets, TLI, UNIX FIFOs and STREAM pipes, and Win32 Named Pipes. In addition, the ACE C++ wrappers encapsulate the OS filesystem APIs.
ACE employs a number of techniques to minimize or eliminate the performance overhead.

- Memory management components. The ACE memory management components provide a flexible and extensible abstraction for managing dynamic allocation and deallocation of interprocess shared memory and intraprocess heap memory.

The C++ wrappers provide many of the same features as the OS adaptation layer in ACE. However, these features are structured in terms of C++ classes and objects, rather than standalone C functions. This OO packaging helps to reduce the effort required to learn and use ACE correctly [15].

For instance, the use of C++ improves application robustness because the C++ wrappers are strongly typed. Therefore, compilers can detect type system violations at compile time rather than at runtime. In contrast, it is not possible to detect type system violations for C-level OS APIs, such as sockets or filesystem I/O, until runtime.

ACE employs a number of techniques to minimize or eliminate the performance overhead. For instance, ACE uses C++ inlining extensively to eliminate method call overhead that would otherwise be incurred from the additional type safety and levels of abstraction provided by its OS adaptation layer and the C++ wrappers. In addition, ACE avoids the use of virtual methods for performance-critical wrappers, such as send() / recv() methods for socket and file I/O.

The ACE Framework Components

The remaining ~40% of ACE consists of communication software framework components that integrate and enhance the C++ wrappers. These framework components support the flexible configuration of concurrent communication applications and services [8]. The framework layer in ACE contains the following components:


- Service initialization components. The ACE Connector and Acceptor components [3] decouple the active and passive initialization roles, respectively, from application-specific tasks that communication services perform once initialization is complete.

- Service configuration components. The ACE Service Configurator [9] supports the configuration of applications whose services may be assembled dynamically at installation time and/or runtime.

- Hierarchically layered stream components. The ACE Streams components [1, 8] simplify the development of communication software applications, such as user-level protocol stacks, that are composed of hierarchically layered services.

- ORB adapter components. ACE can be integrated seamlessly with single-threaded and multithreaded CORBA implementations via its ORB adapters [16].

The ACE framework components facilitate the development of communication software that can be updated and extended without the need to modify, recompile, relink, or often restart running applications [8]. This flexibility is achieved in ACE by combining (1) C++ language features, such as templates, inheritance, and dynamic binding, (2) design patterns, such as Abstract Factory, Strategy, and Service Configurator [9, 17], and (3) OS mechanisms, such as dynamic linking and multithreading.

Self-contained Distributed Service Components

In addition to its C++ wrappers and framework components, ACE provides a standard library of distributed services that are packaged as self-contained components.
Although these service components are not strictly part of the ACE framework library, they play two important roles:

1. Factoring out reusable distributed application building blocks. These service components provide reusable implementations of common distributed application tasks such as naming, event routing [2], logging, time synchronization, and network locking.

2. Demonstrating common use-cases of ACE components. The distributed services also demonstrate how ACE components like Reactors, Service Configurators, Acceptors and Connectors, Active Objects, and IPC wrappers can be used effectively to develop flexible, efficient, and reliable communication software.

Higher-level Distributed Computing Middleware Components

Developing robust, extensible, and efficient communication applications is challenging, even when using a communication framework like ACE. In particular, developers must still master a number of complex OS and communication concepts such as:

- network addressing and service identification
- presentation conversions, such as encryption, compression, and network byte-ordering conversions between heterogeneous end-systems with alternative processor byte-orderings
- process and thread creation and synchronization
- system call and library routine interfaces to local and remote interprocess communication (IPC) mechanisms

It is possible to alleviate some of the complexity of developing communication applications by employing higher-level distributed computing middleware, such as CORBA [18], DCOM [19], or Java RMI [20]. Higher-level distributed computing middleware resides between clients and servers and automates many tedious and error-prone aspects of distributed application development, including:

- authentication, authorization, and data security
- service location and binding
- service registration and activation
- demultiplexing and dispatching in response to events
- implementing message framing atop bytestream-oriented communication protocols like TCP
- presentation conversion issues involving network byte-ordering and parameter marshalling

To provide developers of communication software with these features, several higher-level middleware applications are bundled with the ACE release. The ACE ORB (TAO) [21] is a realtime implementation of CORBA built using the framework components and patterns provided by ACE. TAO contains the network interface, OS, communication protocol, and CORBA middleware components and features shown in Figure 2. TAO is based on the standard OMG CORBA reference model [18], with the enhancements designed to overcome the shortcomings of conventional ORBs [22] for high-performance and realtime applications. TAO, like ACE, is freely available at <www.cs.wustl.edu/~schmidt/TAO.html>.
JAWS [23] is a high-performance, adaptive Web server built using the framework components and patterns provided by ACE. Figure 3 illustrates the major structural components and design patterns in JAWS. JAWS is structured as a framework of frameworks. The overall JAWS framework contains the following components and frameworks: an Event Dispatcher, Concurrency Strategy, I/O Strategy, Protocol Pipeline, Protocol Handlers, and Cached Virtual Filesystem. Each framework is structured as a set of collaborating objects implemented by combining and extending components in ACE. JAWS is also freely available at <www.cs.wustl.edu/~jkh/research/>.

Lessons Learned Developing and Deploying ACE
This section summarizes the lessons I’ve learned during the past seven years developing the reusable OO communication software components in the ACE framework and deploying ACE in a wide range of commercial applications in the avionics, telecommunications, and medical domains.

Software Reuse Fails Largely for Nontechnical Reasons
In theory, organizations recognize the importance of reuse as a means to reduce cycle-time and improve software quality. In practice, many factors conspire to make it hard to achieve systematic software reuse. Most of the impediments are largely political, economical, organizational, and psychological, rather than technical. For instance, teams that develop reusable middleware platforms are often viewed with suspicion by applica-
tion development teams, whose members resent the fact that they are no longer empowered to make key architectural decisions.

**Successful Reuse-in-the-large Requires Prerequisites**

In my experience, large-scale reuse of software works best when the following conditions apply:

- The marketplace is highly competitive. In a competitive environment, time-to-market is crucial. Therefore, it is essential to leverage existing software to substantially reduce development effort and cycle time. When a marketplace is not competitive, however, there is often a tendency to reinvent rather than reuse.

- The application domain is challenging. Components that are relatively easy to develop, such as generic linked lists, stacks, or queues, are often rewritten from scratch, rather than reused. In contrast, developers are generally willing to reuse highly complex components, such as dynamic scheduling frameworks [24] or realtime ORBs [21], because building complete solutions from scratch is too difficult, costly, and time-consuming.

- The corporate culture is supportive. It is hard to develop high-quality reusable components and frameworks. In particular, it is hard to reap the benefits of reuse immediately. A great deal of effort must be expended initially to produce efficient, flexible, and well-documented reusable software artifacts. Thus, an organization must support an effective process in order for reuse to flourish. For instance, developers must be rewarded, not punished, for taking the time to build robust reusable components. Moreover, the reuse process must reward production of concrete software artifacts, rather than endless abstract metamodels or high-level design documents.

These prerequisites often do not exist in contemporary organizations. In such cases, I've observed that organizations often fall victim to the “not-invented-here” syndrome and redevelop most of their software components from scratch. Unfortunately, increasing deregulation and global competition make it hard to succeed with this type of development process.

**Iteration and Incremental Growth Are Essential**

It is crucial for organizations to explicitly recognize that good components, frameworks, and software architectures require time to craft, hone, and apply. In general, developing, using, and reusing software requires a mature organization that can distinguish key sources of variability and commonality in its application domain. Identifying and separating these concerns require multiple iterations.

For reuse to succeed in-the-large, management must have the vision and resolve to support the incremental evolution of reusable software. Fred Brooks's observation that “Plan to throw the first one away, you will anyway,” [25] applies as much today as it did 20 years ago. Moreover, in my experience, “the best is often the enemy of the good” when it comes to deploying reusable software frameworks and components. Often, an 80% solution that can be deployed and evolved incrementally is preferable to waiting for a 100% solution that never ships.

**There's No Substitute for Hands-on Experience**

Developing high-quality communication software is hard; developing high-quality reusable communication software is even harder. The principles, methods, and skills required to develop reusable software simply cannot be learned by generalities. Instead, developers must learn through hands-on experience how to design, implement, opti-
mize, validate, maintain, and enhance reusable software components and frameworks. Only by actively engaging in these activities will developers truly internalize good development practices and patterns.

**Integrate Infrastructure Developers with Application Developers**

Most useful components and frameworks originate from solving real problems in a particular application domain, such as telecommunications, medical imaging, avionics, and Web programming. A time-honored way of producing effective reusable components, therefore, is to generalize them from working systems and applications. This was how ACE evolved.

I’ve found that creating “component teams,” which build reusable frameworks in isolation from application teams, is often counterproductive. When intimate feedback from application developers is lacking, the software artifacts produced by component teams rarely solve real problems and are unlikely to be reused systematically.

**Design to an Architecture Rather Than Program to a Particular Middleware Technology “Standard”**

It is very risky to expect that emerging industry middleware standards, like CORBA, DCOM, or Java RMI, will automatically eliminate the complexity of developing communication software. No single solution is a panacea, nor are “standards” necessarily ubiquitous or implemented consistently.

Therefore, for complex communication software systems, it is essential to design and use architectures that can transcend any specific middleware technology standard. I’ve found it is much more effective to devise a common software architecture that can be instantiated on multiple middleware platforms, rather than programming directly to a particular middleware API, which can rapidly become obsolete.

**OS API “Wars” Are Largely Irrelevant**

ACE’s OS adaptation layer makes the selection of the native OS API, e.g., POSIX vs. Win32 vs. realtime operating systems, largely an implementation detail. Using ACE, it is straightforward to develop highly portable communication software that runs efficiently on a wide range of operating systems and C++ compilers. Moreover, ACE provides this portability without incurring the performance penalties associated with interpreted virtual machines. (However, a Java version [26] of many ACE components is also available at <www.cs.wustl.edu/~schmidt/ACE.html>.) Thus, the portability provided by ACE allows developers to select an OS platform based on features, price, performance, development tools, and ease of integration with other applications.

**Beware of Simple(minded) Solutions to Complex Software Problems**

Trying to apply overly simple solutions to complex problems is an exercise in frustration and a recipe for failure. For instance, attempting to translate the software implementations entirely from high-level SDL specifications or “analysis rules” rarely succeeds for complex communication systems. Likewise, using trendy OO design methodologies, modelling notations, and programming languages is no guarantee of success. In my experience, there’s simply no substitute for employing skilled software developers, which leads to the following final “lesson learned.”

**Respect and Reward Quality Developers and Architects**

Ultimately, reusable components and frameworks are only as good as the people who build and use them. Developing robust, efficient, and reusable middleware requires
teams with a wide range of skills. We need expert analysts and designers who have mastered design patterns, software architectures, and communication protocols to alleviate the inherent and accidental complexities of communication software. Moreover, we need expert programmers who can implement these patterns, architectures, and protocols in reusable frameworks and components.

In my experience, it is exceptionally hard to find high-quality software developers. Ironically, many companies treat their developers as interchangeable, "unskilled labor" who can be replaced easily. Over time, companies that respect and reward their high-quality software developers are increasingly outperforming those that do not.

Concluding Remarks
Computing power and network bandwidth have increased dramatically over the past decade. However, the design and implementation of communication software remain expensive and error-prone. Much of the cost and effort stems from the continual rediscovery and reinvention of fundamental patterns and framework components across the software industry. However, the growing heterogeneity of hardware architectures, the diversity of OS and network platforms, and global competition make it increasingly costly to build correct, portable, and efficient applications from scratch.

Object-oriented application frameworks and patterns help to reduce the cost and improve the quality of software by leveraging proven software designs and implementations to produce reusable components that can be customized to meet new application requirements. The ACE framework described in this article illustrates how the development of communication software like ORBs and Web servers can be significantly simplified and unified.

The widespread adoption of ACE is a testament to the power of an open source software process and to the benefits of systematic software reuse in complex communication systems. One key to the success of ACE has been its ability to capture common communication software design patterns and consolidate these patterns into flexible framework components. The framework components efficiently encapsulate and enhance low-level OS mechanisms for interprocess communication, event demultiplexing, dynamic configuration, concurrency, synchronization, and filesystem access.

The ACE C++ wrappers, framework components, distributed services, and higher-level distributed computing middleware components described in this article are freely available at <www.cs.wustl.edu/~schmidt/ACE.html>. This URL contains complete source code, documentation, and example applications, including JAWS and TAO.

ACE has been used in research and development projects at many universities and companies. For instance, ACE has been used to build realtime avionics systems at Boeing [27]; telecommunication systems at Bellcore [4], Ericsson [28], Motorola [2], and Lucent; medical imaging systems at Siemens [9] and Kodak [16]; and distributed simulation systems at SAIC/DARPA. It is also widely used for research projects and classroom instruction.

A description of many of the projects using the ACE, TAO, and JAWS frameworks is available at <www.cs.wustl.edu/~schmidt/ACE-users.html>. In addition, <comp.soft-sys.ace> is a USENET newsgroup devoted to ACE-related topics.

Acknowledgments
Much of the success of ACE is due to the dedication of the core development team at Washington University, as well as the hundreds of developers throughout the Internet.
who contribute their time and effort to improve ACE. I greatly appreciate their help, as well as the support of USENIX, which has sponsored some of the research on TAO's realtime scheduling service.

Notes


[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995.


a Tcl story

One of our customers wanted a solution that would achieve batch processing of SGML documents having a database updated to trace the documents’ status.

Like other word processors, WordPerfect can do a lot for you in text handling, and it can also process SGML documents – from creation to printed outputs according to style sheets.

WordPerfect integrates a macrolanguage that can be used to customize its behavior (even to build dialog boxes). But in this case, our needs weren’t limited to document processing: they included the ability to access an Informix database in order to know which documents to handle and the ability to update it.

WP’s macrolanguage is not aimed at addressing such tasks without extension. Rather than investing in building an extension to a proprietary language (which could be used only within WordPerfect), we took a look at alternative solutions.

Because of WordPerfect’s OLE server capabilities, it was obvious to us that our solution could be based on another language that would better meet our needs. Tcl seemed to be a good option. One attractive aspect of Tcl is its high integration capability with other applications. Furthermore, we use it to process SGML documents, activating the Web pages, etc. Our glue was found.

To gain the benefits of speed and better Win platform integration, we chose Tcl 8.1a1.

After designing our application’s basis, we looked at the two things that were not built into Tcl: database connection and WordPerfect handling.

To remain independent of the database, we decided to work with ODBC. After searching the Web for database tools and testing several ODBC extensions for Tcl, we chose TclODBC15 from Roy Nurmi <Roy.Nurmi@jw.lu>: retrieving, inserting, and updating data worked fine. It’s a great piece of work.

Unfortunately, a problem showed up. In Europe, we use a lot of characters like the ones with accents, and strings with such characters coming out of the database were not string compatible with Tcl’s strings. After we got in touch with Roy about that, he experienced these problems, too. TclODBC15 was designed and tested on Tcl 7.6 and not on Tcl 8.1, which has special string handling based on UNICODE.

Roy decided to rewrite/update TclODBC to meet Tcl 8.1 needs, i.e., supporting the UNICODE handling. In the meantime, we found a way to work around this problem. I spent some time exploring the Tcl 8.1 source, especially the parts on string handling. I found that the scan instruction could convert our strings from ODBC to the Tcl 8.1 format, making the string comparison work well: the DB solution was up and ready to work.

Now it was time to work on the WordPerfect side. At first we thought that DDE calls would be powerful enough to have our Tcl applications deal directly with WordPerfect. But DDE calls were suffering some lack of power and stability. We decided to see if it was possible to bind Tcl and WordPerfect with OLE.

After looking at the different extensions available for Tcl, we realized that nothing was available to do what we needed to do. There were extensions for visual objects like widgets, but nothing that would allow us to control WordPerfect. Was it time for us to try to build an extension for Tcl on a Win platform and do it from scratch? We gave ourselves a week to try to get it done!
The first thing was to see just how to build an extension on a Win platform for Tcl. This was relatively easy because there are a lot of examples on the net. We decided to build it like a new instruction with several available options.

The principle was simple: after being loaded, the extension creates a new instruction that is registered by the Tcl core. Then each time the instruction is used a special procedure tests the option, seen as parameters to the instruction, to see what secondary procedures should be called.

Then we needed to build an interface for WordPerfect’s OLE server. This was our main problem. Generally, working with OLE servers is achieved by using a C++ interface that hides the complexity of OLE technology. But we wanted to have access to these inside mechanisms: this drove us to spend the largest part of our efforts on this.

The binding is rather primitive: we instantiate our object and then have the ability to call its methods by using the Tcl instruction’s options. Only the most needed methods are built inside the extension (e.g., opening a file, printing it, starting a WordPerfect macro). But the others are also accessible, through a generic method caller.

One of the problems we encountered then was to map the Tcl types to WordPerfect’s methods parameters. Fortunately, these methods were using only two types of parameters: string and long int. When WordPerfect is called from Tcl, it acts in a proprietary way. For example, you may not quit the application from inside its menu; only Tcl script that called it can do that. For the best results, you would have to use the extension under the Tcl graphical interpreter WISH. This is needed so that the Win events are parsed correctly by Windows.

After testing our new extension and getting WordPerfect to open documents, run some macros, and then printing them (all driven by Tcl and logged in our database), we needed to conduct some more tests to establish whether it was stable and usable.

At that moment, we thought it could become an Open Software to be contributed to the Tcl community and thus gain sufficient feedback to initiate refinement cycles if so requested. To do this, we have made the extension available on the net under GNU license and opened a discussion group on one of our Web servers.

After the announcement in the Tcl newsgroup, the extension was downloaded about ten times, but we were only partially successful in collecting feedback. Nevertheless, there was a common request for a more general extension based on the OLE principle that would open Tcl to a great number of applications based on OLE.

This would be a nice feature to be added in the future, but for a proper implementation more people should be involved. If there are volunteers, we would be glad to coordinate the work and participate actively. Co-workers are welcome to contact us (email: <gb@mail.jwaylu>).

In conclusion, Roy Nurmi has recently completed his new version of TclODBC2.0, completely integrated with Tcl8.1. It should be available soon on the net <Roy.Nurmi@iki.fi>.
Agenda: a reminder tool

Agenda is a small, multiplatform tool with one purpose: to inform the user about scheduled events. How many times have you forgotten a friend’s birthday or anniversary or some exhibition you wanted so much to see? With Agenda you merely enter the event description, date, the number of days you want to be informed before the event, and some other things, and that’s it. Just turn on the computer and you’ll be told all about it.

Agenda is divided in two scripts written in Tcl/Tk. The availability of this language on a wide variety of platforms makes it possible to run this application in many environments. For more information on Tcl/Tk refer to Scriptics home page at: <http://www.scriptics.com> or The Tcl/Tk Consortium at <http://www.tcltk.org>.

Downloading, Installing and Configuring Agenda

To download Agenda go to <http://www.geocities.com/SiliconValley/6333/agenda>. This is the official, and only, site. Download the files Readme, agenda, and agenda_chk. A brief description of each follows:

- Readme (text file). This contains installation procedures and other information.
- agenda (Tcl/Tk script). This script is the editor of events. You can add, remove, or edit events written in the event file.
- agenda_chk (Tcl/Tk script). This script will read the event file and notify you about the events that are going to happen.

To install and configure Agenda follow these three steps:

Step 1

Change the parameters in the first lines of the scripts. In the script agenda:

EVENT_FILE: the path and name of the text file where the events are to be stored. In a UNIX environment this should be something like ~/.events. This is the default. If you are a no-UNIX user, this will probably not work. For example, a Windows user might change it to c:\data\events.

In the script agenda_chk:

EVENT_FILE: should be exactly the same as in agenda
WIDTH: number of columns in the window list of events presented by this script
POS_X, POS_Y: the position on screen of the window list

Step 2

Copy the modified scripts to some directory in your path.

Step 3

Edit the startup procedure of your machine to call agenda_chk. This way your machine will inform you of the schedule at startup. You can call agenda_chk any other time.

Note that the first line of each script has a reference to the wish interpreter. Though usually installed in /usr/local, this might not be the case. If not, you have to change this line appropriately.
Working with Agenda

Now we are ready to go. Let’s start by calling the Agenda editor. You should see a window with two buttons on top, about and quit, and four buttons on the bottom, add, edit, mark/unmark for removal, and write. In the middle is the list of events in the event file. It should be empty now.

Let’s add an event. Pressing add will pop up a window requesting the event data:

? Event: the name of the event. For our tour type “Ana’s Anniversary.”

? Day: the day of the event. Let’s assume it’s February 12. Type 12.

? Month: the month: Type 2

? Year: Any year should do. Type an asterisk (symbol *). The asterisk can be used in the day and month fields, too. When agenda_chk finds this symbol, it replaces it with the current day/month or year.

? Before: number of days before the event you should be notified. For example, if you type 5, then you are notified from day 7 (12–5) on. (This gives you time to buy Ana her present.)

? After: number of days after the event you should still be notified. If you type 2, you will be notified until day 14. (In case you don’t turn on the computer until the 14th, you can always call Ana and make up a tale.)

? Action: this field has three flags: Cyclic, Once, and Don’t show.

? Cyclic: if flag is on, then after the event has been posted, it is left in the event file. If off, this event is deleted from the event file and appended to the event log. In our case it should be on.

? Once: if on, you are notified only once during the valid period, that is between days day-before and day+after. If off, you are notified every time in that period. Just leave it off.

? Don’t show: if on, this event will not be shown. This flag is turned on after the first notification when Once is on.

? Command: command to run. Let’s say you have a playmidi program and a birth-day.wav sound file. Type “playmidi birthday.wav”. This can be very useful for periodic tasks like backups or other system maintenance.

? Comments: any comments needed. Just type “Buy her something nice.”

Because the separator used in the event file is the comma, be sure never to use it in the fields described above. Press OK to add the event. It should appear in the list. As an exercise, add two or three more events that should be noted today. To edit, just press the button. To delete, mark it as deleted (D).

Finally, press the write button to actually write the list to the event file.

Now let’s call agenda_chk to check the events. If any of those extra events you supposedly added is correct, you should have something on the list. Events with an * at the beginning have extra comments. Double click on them to see the comments.
The File Format
The format is as follows:


In the action field we can find the letters c, o, or x if the flags

cyclic, once, or don’t show are on respectively. Should you want to
write more about an event, just terminate the line with a backslash
and continue on the following line. Several lines are permitted,
just terminate them with the backslash. Note: use a \ only if there
is another line to come. Some examples

follow:
10:6:*:*:0:0:c:Portugal Day
15:*:*:0:10:co:backup_data:Data Backup
3:12:1990:0:4:>:Contact Mike \
Discuss the house problem \
Don’t forget to mention the payment method

Some Interesting Events
Next are some ideas worth looking at:

*:*:*:0:0:c:show_joke random:Joke a Day
1:*:*:5:0:c:backup:BACKUP TIME \
mail boxes \ mydata directory \ server configuration
1:*:*:0:10:co:scanvirus alldrives:Checking for VIRUS
12:*:*:0:0:c:Tomorrow is 13th. Lookout for VIRUS

The Future of Agenda
The future will be decided mainly by the acceptance of this application. If I have
enough encouragement, I will try to work on it some more. If you like this application
and find it useful, let me know. Send me mail with some nice words. Contributions are
welcome. If you have new ideas for Agenda, more features, bug reports, etc., send them
in. Another way to contribute is by building lists of events and distributing them. For
example you could compile all the social events of your city and distribute it. This
could boost your social life.
managing open source software

In 1991, I was bitten by the Perl bug while working for a large Japanese computer manufacturer. I wanted to use Perl in our production code, but was immediately beset by two obstacles.

The first obstacle was management and the arguments regarding free software so prevalent at the time. You have heard them all before. How could something free be any good? We don't want software from some dope-smoking college student. Where's the sales rep to take us out to lunch? Where do we get support?

Things sat for a while until the Camel book was published. Thus emboldened, I took my copy to our VP, seeking permission once again. This time I was successful. He immediately understood what Larry was doing: “Ha Ha. Give away some software; clean up on the book end. What a bunch of suckers. I like this guy!”

This was not quite the reasoning I had intended to use, but I let it go at that. “Yes, sir, I think you’ve got him pegged. No use trying to pull the wool over your eyes!” [1] The same thing happened when I started using Tcl/Tk. Upon publication of his book, John Ousterhout went from “suspected college radical” to “cunning entrepreneur” [2].

Once I had the blessing of the powers that be, I hit the second obstacle: managing the configuration and deployment of our open source software. My first tries were rather haphazard -- variations on tarring /usr/local and putting it on the distribution tape (quite a mistake, as we will see). I learned a lot along the way. I hope I can help you avoid mistakes and deploy your open source solutions with a minimum of fuss and trouble.

Although I emphasize the case where you need to ship some of the open source packages as part of your own product or install onto a computer that is not part of your local network, the process is still useful even if you are just installing the software locally. Ignore the steps that don’t apply to you.

Getting Started

The first thing to do is to decide on the directory structure that will work best for your project. Several guidelines make this easier.

Don't install the software into /usr/local. This will work fine for your site, but when you try to install the software at the customer site, you will probably conflict with things the customer has already placed there.

Instead, choose an installation prefix that will be unique to your organization. Get permission from your customers to create that directory on their system. Nobody else will install software there, so you don’t need to worry about version conflicts, etc. As an example, we use the prefix /aitools so that executables are installed in /aitools/bin.

Build from a directory that is not a subdirectory of your installation directory. I initially made the mistake of building in /aitools/src. I wanted to test my installation procedure on a clean directory. Having the source code, etc., sitting there made this a lot more difficult because I was naturally reluctant to delete it regularly.

Don’t put anything in the installation directory besides the software you wish to ship to the customer site. I installed emacs and the other usual tools there and immediately had a huge directory that would have used up all the disk space on the customer system.
For the rest of this article, I will continue to use this directory structure as an example:

/aitools – where the files are installed. In autoconf terms, --prefix=/aitools.

/aitools2 – where our local tools are installed and where all of our source is configured and built.

/aitools2/src – where the source files are configured and compiled. These directories are version controlled.

/aitools2/dist – where the original distribution and patch files are stored.

/aitools2/cvs – we use CVS for version control. This is the repository for the files in /aitools2/src.

/aitools2/bin – where our local tools are installed.

Source Control
Most open source software packages are distributed as tar files, usually with the version number as part of the filename. All downloaded distributions and patches are stored in /aitools2/dist.

I untar these files under /aitools2/src and import them into the source repository. It is usually easier to configure the software before importing. If the configuration process creates any important files, it will save you the trouble of having to add them to the repository separately.

Of course, you will commit your changes to the repository whenever you apply a patch. This is one area where the patch program and CVS work very well together. Performing the following steps will make it very easy to keep track of the patches that have been applied:

    cvs update # make sure files are up to date
    patch < ../.. /dist/some-patch-file
    cvs commit -m "patched from file ../.. /dist/some-patch-file"

Of course, any local modifications you make will be similarly tracked. If you fix a problem, don’t forget to send a note to the package maintainer!

If you use a checkout-based version control system, you can either inspect the patch file to see which files will be affected or check out everything and release the checkout for all the files that are not modified by the patch. On a related note, if you use a file-system-based version control system such as Clearcase, you may need to scan the build procedure and replace commands that affect directory entries (such as chmod) with the appropriate commands from the version control system (such as cleartool chprot –chmod).

In general, I don’t recommend overlaying major distribution versions on top of each other; even though this is how the package maintainers keep the files. Unless you are seriously maintaining the package yourself, it seems to be easier to consider each release of the package as an independent product. Using Tcl as an example, your directory structure would look like this:

/aitools/src/
    tcl17.5/
    tcl17.6/
    tcl18.0/
Configuration and Building
One of the most frequently made mistakes during the configuration process is not reading the configuration instructions. Be sure and do this, paying special attention to the specification of the installation area. When you configure the software, specify /aitools as the prefix for the installation area.

After the software is configured, build and test it according to the package instructions. If you are going to be creating packages, you should take before and after snapshots of the installation area to assist in making the list of files to be included in the package.

The majority of open source software currently uses GNU autoconf (a wonderful system!) to configure the package. The steps discussed so far for a typical autoconf-based package would go like this:

First, we unpack and configure the source code.
- download the tar file into /aitools/dist
- cd /aitools/src
- tar xvf ../dist/package1.0.tar
- cd package1.0
- ./configure --prefix=/aitools

Next, we need to put the package under version control. With CVS, we usually import the package, delete the source tree, and then check the package out. This results in a properly controlled CVS directory.
- cvs import package1.0
- cd ..
- rm -rf package1.0
- cvs co package1.0

Next, we build and test the package.
- cd package1.0
- make all
- make test

And finally, install the software into /aitools. We take the chance to make a snapshot of the installed files at this time.
- find /aitools -type f -print >files-before-install
- make install
- find /aitools -type f -print >files-after-install
- create package

Quality Assurance and Project Management
If you are preparing your open source software for shipment to a customer, you will need to take into account your company's QA and Project Management groups.

Your QA group probably has some standard procedures for documenting and executing test cases on your company's products. If so, this is a chance to let your open source software shine. By the very nature of open source development (source releases,
multiple platforms, many people building and testing the software), most open source projects end up with an impressively large collection of test cases. Many of these are in the form of automatically verifiable regression tests.

So take the opportunity to write up a test document using your company's standard format. The sheer volume of tests (for Tcl/Tk, nearly 10,000 combined tests) can garner quite a few supporters for your favorite software.

Likewise, your project management group probably has a standard scheme for assigning product codes and the like. Follow up with this, and get each piece of software its own product code, project number, or whatever it is your company uses to track its software. It may seem like a bureaucratic headache, but cooperating in this area makes your open source software seem a lot more "normal."

Building the Packages

Once the software is built and installed, you need to create an installation package. There are several options for creating the package, but the one I like best these days is RPM, the Redhat Package Manager. Originally written to manage Redhat Linux, it now runs on all major UNIX platforms. I don't have the space to go into all of its features, but you can get more information at <http://www.redhat.org>.

Most software packages install both runtime files (executables, shared libraries, runtime support files, etc.) and development related files (header files, documents, static libraries, demos, etc.).

Depending on your situation, you can create either a package consisting of all the files, both runtime and development, or two separate packages for the runtime and development related files. Which you choose depends on your particular circumstance. Will the customer benefit from having the full installation? Do you have the disk space to spare?

Splitting the packages (if that is what you wish to do) is relatively straightforward. The easiest way to do it is to copy all the files into a temporary location and start deleting all the development files. You will eventually be left with only the executables and runtime support files. When you have this list, take the complement, and that becomes your development list.

Installation Procedures

Now that you have your package(s) built, you are ready to deploy to your customer site. You can include your newly built open source runtime packages with your standard distribution media, and just add a step in your installation procedure to install those prerequisite packages first.

If you use a package manager such as RPM, you can specify that these packages are prerequisites to your own packages and rely on the fact that RPM will make sure the packages are installed in the proper order. Once the packages are installed, you use the standard package maintenance commands to query and verify your installation.

Relocating Packages

In some cases, you might not have the luxury of being able to specify a particular directory where your runtime tools will be installed. If this applies to you, there are three options open to you.

1. Talk to the individual or group placing that restriction on the project and try to convince them that it is OK to prespecify the location of your support tools. Many times it is an internal decision based on the (usually) admirable goal of total software flexibility. If this is so, you can try to make the case for a fixed location based on the
engineering trade-off of reduced development cost and more consistency across customer sites.

2. Most packages allow the user to set environment variables to override the default runtime settings. You can ask your end-users to set these as necessary. I don’t recommend this, however, based on the near zero probability of everyone doing this correctly.

3. You can write a relocation program that will modify the runtime files to point to a new installation area.

Relocating Programs

It is not difficult to relocate packages, although there may be some system dependencies in doing so. The basic procedure is:

- Begin by configuring the package with a long, unique prefix. It should be long enough to provide enough path space for the actual location. It should be a unique string so that it can be searched for in all the files in the package. A good value might be something like:

  /tmp/xyz

  Whatever value you pick, be sure and grep for it in your package to ensure its uniqueness.

- Build, install, and create the package as usual. Upon installation, perform the following two steps:

  1. For all text files in the package, substitute the preconfigured prefix
     (/tmp/xyz) with the real installation area.

  2. For all binary files, do the same thing, but pad all trailing bytes with nul (zero) bytes.

Creating Standalone Programs

Sometimes it is desirable to eliminate all external dependencies on a software package, thereby eliminating the need to ship a runtime system for that package.

If the package you are using is provided as simply a set of library files, you can either build the software with static libraries or include the correct linker flags (such as -lstatic) to cause the linker to include the library in the executable file. You will pay the cost of larger executables and possibly less efficient memory usage, but this may be an acceptable trade-off for your project.

If you are using one of the popular open source languages, you may be able to use the special features of that language to eliminate the need for shipping the runtime files normally associated with that language.

Two examples of this should suffice. If you are using Python, you can use freeze to create a C file that, when compiled and linked, will produce a standalone executable with no external dependencies. If you are using Tcl, you can use tc12c, one of the components of the Plus Patches, to do the same thing. Similar features exist for most of the common languages. Consult your documentation or resident language guru for details.

Summary

There are countless examples where open source software has made a significant positive impact on commercial development projects. Proper source and project management is a critical component in making your open source efforts a success.
using Python

If you’re like most developers, one of the first questions you may ask when you hear about a new programming language or tool is: what can I use it for? After all, programming languages are tools; although they can be interesting to study in isolation, their real utility lies in the context of applications.

**Python Defined**

In terms of acronyms, Python can be classified as both a VHLL (very high level language) and an OODL (object-oriented dynamic language). Although this puts Python in the same general category as languages such as Smalltalk and Eiffel, Python has a flavor all its own. For instance, Python has a very practical orientation: unlike some OO (object-oriented) languages, Python was created by and for engineers interested in solving day-to-day programming tasks.

Because Python is a general-purpose language, arriving at a single all-encompassing definition isn’t an easy task. What Python is depends much on how it is used. To some, Python serves as programmable front-end to libraries coded in a compiled language like C or C++. For others, it takes the form of an embedded scripting language for customizing larger systems. And many programmers use Python as a standalone language, leveraging its library of pre-coded system interfaces. But in terms of basic functionality, Python can be described in a variety of ways.

**Object-oriented Scripting Language**

Python is a fundamentally object-oriented language. Its “class” system supports modern OO development paradigms and makes Python ideal for use in OO systems. Perhaps more important for a scripting language, Python’s object model is surprisingly easy to use, yet it supports advanced OO concepts such as operator overloading, multiple inheritance, and more.

**Standalone Rapid Development Tool**

Python provides both a conceptually simple language and rapid development-cycle turnaround. It is specifically optimized to support speed of development. In fact, many programmers find Python development to be orders of magnitude faster than other approaches. By combining development speed with integration tools, Python fosters radically faster development modes.

**Next-generation Extension Language**

Like many scripting languages, Python programs can be both extended with modules coded in a compiled language like C and embedded in a C program. But unlike many scripting tools, Python provides both integration with external components and a powerful OO language. Because of this combination of utility, Python can address a wide range of problem domains.

**Freely Available, Interpreted Language**

By most definitions, Python is an interpreted language. More accurately, Python code is compiled to portable byte code that is run by a virtual machine. This scheme provides both fast turnaround after program changes and reasonable execution speed. Finally, Python is true freeware: its source code is freely available, and it may be embedded in products without copyright constraints or fees.
Some Quotable Quotes

But don’t take this writer’s word for it. Some of the comments that have been made by Python programmers on the Internet speak volumes about the language’s design. Although personal reactions vary, the following quotes are representative of the level of enthusiasm Python typically generates in newcomers:

“Python looks like it was designed, not accumulated.” Python sports a refreshingly coherent design. Python’s creator carefully balanced the need to support practical programming needs with the desire to avoid feature glut and complexity. The result is a simple language that scales well to support larger systems. Moreover, Python has a remarkably clear syntax: for many, its inherent readability makes it ideal for programs that may be reused or maintained by others.

“Python bridges the gap between scripting languages and C.” Python’s design and implementation place it somewhere between compiled languages such as C and traditional scripting languages like csh, Awk, and Perl. Python’s syntax resembles languages like C (e.g., there are no “$” variable prefixes), but its high-level programming tools and lack of compile and link steps are familiar to users of other scripting languages. In some sense, Python provides the best of both worlds: it supports scripting and embedding, but doesn’t ask programmers to abandon normal standards of development quality.

“Python is the BASIC of the 90s.” Although Python is certainly being applied in real companies and products, many current Python users fall into the “hobbyist” category: they use Python because they want to, not because they have to. In fact, Python’s popularity is largely due to a strong grassroots following among engineers, similar in spirit to the support BASIC enjoyed in the early days of the PC revolution. Naturally, Python isn’t a new and improved BASIC (it incorporates advanced programming tools such as exceptions, modules, and classes), but it generates similar excitement.

“Python is as easy as or as powerful as you want it to be.” Python programs can range in complexity from simple five-line scripts, to complex object-oriented frameworks. Program complexity can be scaled for both the programmer’s skill level and the problem at hand. For instance, Python supports OOP, but does not impose it: classes are an optional language tool. Programmers (not language designers) decide whether OOP, and other advanced Python tools, are warranted for each task.

“Python: less filling, tastes great.” This quote will probably be more meaningful if you’re familiar with American beer advertisements, but it underscores one of the central concepts in Python. If you’ve used other compiled, strongly typed languages like C or C++, you’ll find that Python eliminates much of the complexity inherent in traditional tools. In fact, some have called Python “executable pseudocode”: because of their simplicity, Python programs can more closely reflect the problems they are designed to solve.

Some of these quotes will make more sense after we examine the language in more detail. For instance, some of Python’s simplicity stems from its high-level built-in object types (lists, dictionaries, etc.) and the absence of type and size declarations in Python (objects are created dynamically and reclaimed automatically when no longer used).

A Python History Lesson

Python was created by Guido van Rossum in Amsterdam around 1990. Before Python, Guido worked on the ABC programming language and the Amoeba distributed operating-system and both of these influenced Python’s design. For instance, Python was
originally conceived as a scripting language for the Amoeba system, and it inherits ABC's usability study-inspired syntax features. But Python adds practical features to ABC, such as a C-like nature and integration with external C components. Python borrows features from a variety of languages, including Modula, C, Icon, and C++.

Despite the proliferation of snake icons and book covers, Python is named after the BBC comedy series “Monty Python’s Flying Circus.” According to Python folklore, Guido viewed various Monty Python videos at the time he was busy inventing a new object-oriented language; the association naturally arose. This legacy tends to foster a generally humorous (and occasionally irreverent) flavor to the Python community. For instance, the standard labels “foo” and “bar” become “spam” and “eggs” in Python examples, and quotes from Monty Python skits pervade the Python culture (and documents like this).

Python was first offered to the public domain in 1991, and a USENET newsgroup devoted to Python first appeared in 1994 (<comp.lang.python>). Python now enjoys a growing international user community, with a strong following in the U.S., Europe, Japan, and Australia. A nonprofit support organization, the Python Software Activity (PSA), has been established to help manage some of its growth. The PSA maintains Python’s Internet sites and helps organize Python conferences. At this writing, the PSA is hosted by the Corporation for National Research Initiatives (CNRI), which also pays Guido’s salary.

**Uses for Python**

Python’s general-purpose nature makes it applicable to more domains than I can list here. But as a sampling, here’s a quick look at some of the most common things people are using Python for today.

**System Utilities (Shell Tools)**

Python’s existing interfaces to operating system tools make it ideal for writing portable system utilities. Among other things, Python supports UNIX tools such as sockets, regular expressions, and POSIX bindings. On MS-Windows, Python also interfaces with COM, OLE, and more.

**Graphical User Interfaces**

Python’s high-level nature and rapid turnaround are well suited to GUI development. It comes with an OO interface to the Tk GUI library, MFC, X11, and more. The Tk interface runs on X Windows, MS-Windows, and the Macintosh, making it a portable GUI solution.

**Internet Scripting**

Python also comes with a full suite of Internet tools. It has support for CGI scripts, Web browser applets, Web agents and crawlers, HTML generation, and more. Precoded library modules support most Internet protocols: FTP, URLs, HTTP, email, etc. On MS-Windows systems, Python can also be in conjunction with ActiveX scripting, and the new JPython system supports Python/Java integration.

**Database Programming**

Object persistence is a standard part of Python: programmers can save entire Python objects to a persistent store and retrieve them later. Moreover, Python offers interfaces to more traditional database systems, such as Oracle, Informix, Sybase, mSQL, PostGres, and ODBC.
Component Integration

Because Python can be integrated with components written in compiled languages like C and C++, many use Python as a glue language, to control or extend application components. Wrapping libraries in a high-level language like Python, makes them easier to use. And embedding Python in a C/C++ program, opens up the system to customization without recompilation.

Rapid Application Development

To Python programs, components written in Python and C look the same. This allows developers to implement in Python first and later move selected components to a compiled language like C. This development mode is often called prototype-and-migrate: when used well, it can leverage both the development speed of Python and the performance advantages of C.

More Specific Domains

Python is also being used in more specific application areas, such as numeric programming, artificial intelligence, image processing, and distributed object systems (e.g., CORBA). Although these are interesting applications, many are really instances of the component integration potential of Python. For example, Python numeric programming is implemented as compiled-language libraries, integrated into Python for ease of use.

What Python Can’t Be Used For

Naturally, no language can be optimized for every development need. Like most interpreted languages, Python sacrifices some performance in order to maximize development speed. Because of that, Python is not the ideal tool for implementation of time-critical components. For instance, compute-intensive image processing libraries are probably best coded in a compiled language such as C, C++, or FORTRAN.

But even in performance-critical domains, we can make use of Python’s development speed. By implementing compute-intensive components in a compiled language and exporting them to Python, we get a system that’s both easy to use and efficient. The numeric programming example mentioned previously is a perfect example of this approach to development.

Python is optimized for speed of development and designed for multiple-language systems; by adding it to the mix, we can make use of a rapid development language for our system interface and infrastructure while simultaneously achieving performance goals. Moreover, Python can be used to prototype demanding systems before investing the extra time needed to code critical components in C.

Compulsory Features List

Another way to understand Python is by listing its major features. Table 1 summarizes some of the things people seem to like best about the language. It’s by no means complete; the point to notice is that Python combines a wide variety of features in a single, relatively simple language. For many, the end result is a remarkably expressive and responsive language, which can actually be fun to use.
<table>
<thead>
<tr>
<th>Features</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>No compile or link steps</td>
<td>Rapid development-cycle turnaround</td>
</tr>
<tr>
<td>No type declarations</td>
<td>Programs are simpler, shorter, more flexible</td>
</tr>
<tr>
<td>Automatic memory management</td>
<td>Garbage collection avoids bookkeeping code</td>
</tr>
<tr>
<td>High-level datatypes and operations</td>
<td>Fast development using built-in object types</td>
</tr>
<tr>
<td>Object-oriented programming</td>
<td>Code structuring and reuse, C++ integration</td>
</tr>
<tr>
<td>Extending and embedding in C</td>
<td>Optimization, customization, system glue</td>
</tr>
<tr>
<td>Classes, modules, exceptions</td>
<td>Modular programming-in-the-large support</td>
</tr>
<tr>
<td>A simple, clear syntax and design</td>
<td>Readability, maintainability, ease of learning</td>
</tr>
<tr>
<td>Dynamic loading of C modules</td>
<td>Simplified extensions, smaller executables</td>
</tr>
<tr>
<td>Dynamic reloading of Python modules</td>
<td>Programs can be modified without stopping</td>
</tr>
<tr>
<td>Universal first class object model</td>
<td>Fewer restrictions and special-case rules</td>
</tr>
<tr>
<td>Interactive, dynamic nature</td>
<td>Incremental testing and development</td>
</tr>
<tr>
<td>Parser/compiler present at runtime</td>
<td>End-user coding, programs can build programs</td>
</tr>
<tr>
<td>Access to interpreter information</td>
<td>Metaprogramming, introspective objects</td>
</tr>
<tr>
<td>Wide portability</td>
<td>Cross-platform systems without ports</td>
</tr>
<tr>
<td>Compilation to portable byte code</td>
<td>Execution speed, protecting source code</td>
</tr>
<tr>
<td>Built-in interfaces to external tools</td>
<td>O/S, GUI, persistence, DBMS, regular expressions, etc.</td>
</tr>
<tr>
<td>True freeware</td>
<td>May be embedded/shipped without copyright restrictions</td>
</tr>
</tbody>
</table>

Table 1: Major Python Features

Note that most of Python's features aren't new by themselves: it's really how Python combines features that make it what it is. For many, it seems to be the right mix of programming language concepts.

Python in the "Real World"

Python is more than a collection of features. It has become a tool of choice for real programmers in real companies around the world. To get a feel for how companies apply Python for real development tasks, here's a sampling of some of the larger organizations in Python's current user base. This list is necessarily incomplete; check Python's Web site <http://www.python.org> for up-to-date information about other companies applying Python.

Surfing Made Easy (Infoseek, Four11)

Many major Internet-related companies use Python in a variety of roles. For instance, Infoseek was an early adopter of Python in its Web-search systems.

Cockroaches and Pepsi Jingles (Blue Sky Animation)

If you've seen the movie Joe's Apartment, you've seen Python at work. Python has been successfully applied as a scripting language for creating commercial-grade animation.
**Blowing Up the Energizer Bunny (the Alice VR System)**

The University of Virginia uses Python as the scripting interface for its virtual reality system, Alice. One Alice demo features the Energizer Bunny (along with methods to make it explode).

**Rocket Science (Lawrence Livermore Labs, NASA)**

Lawrence Livermore Labs use Python in numerical programming applications and sponsored a recent Python conference. Python is also being applied in a number of ways at NASA.

**Linux Install Tools (Red Hat)**

Any time you install the Red Hat Linux system, you’re using Python: portions of Red Hat’s installation system are written in Python.

**Distributed Object Systems (ILU, Hector)**

ILU and Hector both make use of Python in distributed object programming systems. ILU, from Xerox, is a CORBA-based system that provides bindings to Python’s object model.

**Finding the Grail, Intelligent Agents (CNRI)**

Besides Python, Guido van Rossum also created an extensible Internet/Web browser called (appropriately enough) Grail. It’s written in Python, and allows Python programs to define *applets* executed in the client (browser), much like Java. CNRI also uses Python in research on intelligent agents for the Internet.

**Apples and Oranges and Bananas and Coconuts**

Python is often compared to other free (and not so free) languages, especially on Internet forums. Although language choice is often a matter of style, a brief comparison to similar tools can help point out both some of Python’s primary distinctions and some of the main reasons people choose to use it. Table 2 summarizes some of the most common arguments of Python proponents.

Naturally, your mileage may vary. Some of the distinctions in Table 2 reflect different design goals. For instance, Perl is optimized for text processing, C++ and Java are generally considered to be systems languages (not scripting tools), and Tcl is targeted more toward simple *glue* tasks than larger systems development.

Programmers matter, too: although Python is deliberately designed to prevent unreadable code, there’s nothing stopping us from writing it if we try. Moreover, having many languages to choose from can only be a good thing. In fact, support for multiple-language systems is one of the central concepts in Python: no language can satisfy all our needs at once.

Programmers must be empowered to pick and choose the best language for each task at hand. Python is a great language to have in your development toolbox, but it should not be the only one.
<table>
<thead>
<tr>
<th>Tool</th>
<th>Python Advantage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl</td>
<td>Power</td>
<td>Python is not just a string processor. It is a full-blown language, with support for modules, OOP, exceptions, and more.</td>
</tr>
<tr>
<td>Perl</td>
<td>Coherence</td>
<td>Many find Python's syntax more readable and maintainable. Some also find Python to be less <em>magical</em>: fewer special variables, etc.</td>
</tr>
<tr>
<td>Java</td>
<td>Simplicity, turnaround</td>
<td>Python's built-in objects, dynamic typing, and other features make it much simpler. Moreover, Python can be freely shipped with products.</td>
</tr>
<tr>
<td>C++</td>
<td>Simplicity, turnaround</td>
<td>Because Python is interpreted, it provides much faster turnaround. Further, Python avoids most of the complexity inherent in C++.</td>
</tr>
<tr>
<td>Smalltalk</td>
<td>Conventionality</td>
<td>In Python, “if” statements are not message-receiver objects: Python has a more conventional programming model.</td>
</tr>
<tr>
<td>Scheme</td>
<td>Conventionality</td>
<td>Python’s syntax is closer to traditional languages like C and Pascal. This can be especially important for end-user coding scenarios.</td>
</tr>
</tbody>
</table>

Table 2: Python versus similar tools

For More Information
To learn more about Python, visit Python’s Web site <http://www.python.org>, the author’s Web site <http://www.rmi.net/~lutz>, or the Python USENET newsgroup <comp.lang.python>.

Besides a vigorous online community, you’ll find tutorials, reference manuals, and Python books. For instance, at this writing, O’Reilly <http://www.oreilly.com> publishes two Python books, with a third on the way.
Rhinos in a group are called a crash.

Kangaroos are called a mob. Here at O'Reilly we call our collected books solutions. Why? Because we answer the tough technical questions that our readers are asking—not the ones they've already answered for themselves.

Our readers know they can depend on us for practical, authoritative solutions to the most pressing technical issues of the day. And you should know that you can depend on us for affordable prices and books that not only respond to current issues on the technological front, but help define the issues of tomorrow.

Technical professionals come back to O'Reilly again and again because they know that O'Reilly solutions represent a lot more than a bunch of cute animals. For a fresh, unbiased point of view, turn to O'Reilly. We'll give you what you need—a different kind of animal.
Announcement and Call for Papers

3rd USENIX Windows NT Symposium
Sponsored by USENIX, the Advanced Computing Systems Association

July 12-14, 1999
Westin Hotel
Seattle, Washington

Symposium Web Site: http://www.usenix.org/events/usenix-nt99

Important Dates
Technical paper submission deadline: February 23, 1999
Notification of acceptance: March 23, 1999
Final papers due: June 1, 1999

Overview
Following the tradition established by two very successful symposia, the third Windows NT Symposium continues to provide a forum for the discussion of research and advanced engineering use of Windows NT. The symposium brings together 400 professionals from academic and industrial backgrounds who are actively involved in using or planning to use Windows NT and want to discuss ideas and share information, experiences, and results.

The symposium will include technical presentations of refereed papers, invited talks, informal Demo/Poster and Birds-of-a-Feather sessions, and tutorials. While proceedings will be published, the primary purpose of the symposium is to facilitate two days of useful interaction among the participants.

The symposium will be followed by a one-day, limited-attendance Advanced Research on Windows NT Workshop, where active researchers can share cutting-edge results and discuss the state of the art of Windows NT-specific research.

Topics
Papers that present research results, analyze problem areas, draw important conclusions from practical experience, or facilitate discussion are especially welcome. In addition to experience-centered papers, we solicit papers on a wide range of topics, including but certainly not limited to:

- **Applications**: Development and deployment of large applications environments
- **Manageability**: The use of various models, strategies and tools to address large scale Windows NT or mixed Windows NT networks, and large clusters of Windows NT machines.
- **Security**: Extending the Windows NT security model, and using Windows NT in highly secure environments.
- **Availability**: Experience and research into deploying mission critical applications on Windows NT.
- **Performance and scalability**: Pushing Windows NT to the limit. What is the limit? How to improve it?
- **Networking and distributed systems**: Experiences exploiting the Windows NT distributed technology and the design and use of new networking and distributed services.
- **File and database systems**: Exploiting the I/O systems' advanced functionality.
- **Graphics**: Using the OpenGL environment on Windows NT. Deploying traditional high-performance graphics applications on Windows NT.
- **User interfaces**: Developing new user interface paradigms for Windows NT.
- **Hardware architectures**: The impact of hardware on NT OS software development advances in HAL development.
- **Programming environments**: Programming environments to exploit the wealth of Windows NT functionality.
- **Tools and utilities**: How to make Windows NT a highly productive system.
- **Porting and integration into existing environments**: The cost of porting vs. rewriting, tools, strategies, and trade-offs.

Symposium Organizers
**Symposium Co-Chairs**
Werner Vogels, Cornell University
Stephen Walli, Softway Systems, Inc.

**Symposium Steering Committee**
Michael B. Jones, Microsoft Research
Andrew Hume, Bell Labs
Thorsten von Eicken, Cornell University

**Program Committee**
Brian Bershad, University of Washington
Gary Campbell, Compaq Computers, Inc.
Andrew Chien, University of California, San Diego
Thorsten von Eicken, Cornell University
Jim Gray, Microsoft Research
Michael B. Jones, Microsoft Research
Sam Leffler, Silicon Graphics, Inc.
Richard Oehler, IBM T. J. Watson Research Center
Susan Owicki, InterTrust Technologies Corporation
Karim Petersen, Xerox Palo Alto Research Center
David Steere, Oregon Graduate Institute
Ramu Sunkara, Oracle Corporation
Rumi Zahid, Intel Corporation
Myron Zimmerman, VenturCom, Inc.

**Advanced Workshop Committee**
Todd Needham, Microsoft Research
Werner Vogels, Cornell University
Submitting a Technical Paper
We are soliciting technical papers of concepts, research, and experiences relevant to researchers using Windows NT. We particularly want to encourage researchers who have achieved major results using Windows NT to present them to the symposium. Even if the result itself has been reported elsewhere we encourage you to submit it here as well, although explicitly bringing out and focusing on, rather than downplaying, the role of Windows NT in the research. (Of course, verbatim resubmission of previously published manuscripts is forbidden.)

Note that USENIX conferences, like most conferences and journals, requires that papers not be submitted simultaneously to more than one conference or publication and that submitted papers not be previously or subsequently published elsewhere. Papers accompanied by so-called “non-disclosure agreement” forms are not acceptable and will be returned to the author(s) unread. All submissions are held in the highest confidentiality prior to publication in the Proceedings, both as a matter of policy and in accord with the U.S. Copyright Act of 1976 (Title 17, U.S. Code, Section 102).

Best Paper Awards
Awards will be given for the best paper and best student paper at the conference.

What to Submit
Authors are requested to submit full papers by February 23, 1999.

Submitted papers should be no longer than 10 single spaced 8.5" x 11" pages, including figures, tables, and references. The paper should be submitted through email to usenix-nt-submissions@usenix.org.

The message containing the submission must have a subject line reading “NT symposium submission” and must begin with the following information in this format:

Title: (title of submission)
Authors: (names of all authors)
Contact: (primary contact for submission)
Address: (contact’s full postal address)
Phone: (contact’s telephone number)
Fax: (contact’s fax number if available)
E-mail: (contact’s email address — very important)
Student Authors: (which authors are full-time students)

Submissions must be in Microsoft Word 97, Postscript, or PDF format and should be encoded for transport with either the uunenc or the MIME base64 encoding. Filenames used in your submission should be based on your last name, e.g. smith.doc and smith.html.

Receipt of submissions will be acknowledged by return email within one week; if an acknowledgment is not received, please send email to usenix-nt-questions@usenix.org.

Advanced Research Workshop
Following the symposium, on July 14, a one-day workshop will be held to bring together researchers in intensive sessions to share research results, examine cutting edge performance achievements, and discuss problems and new research directions, all specific to Windows NT. To ensure effective interaction, the workshop is limited to 30-40 participants, and attendance will be by invitation only. Researchers interested in participating are requested to submit an extended abstract describing their work. Details will be made available on the symposium Web site: http://www.usenix.org/events/usenix-nt99/.

Demo and Poster Session
The symposium will include a session where, in an informal setting, participants can present and demonstrate their work. Information on submitting Demo & Poster session proposals will be made available on the symposium Web site: http://www.usenix.org/events/usenix-nt99/.

Tutorials
On July 14, 1999, there will be full-day and half-day tutorials on topics relevant to researchers using Windows NT.

If you are interested in presenting a tutorial at the 3rd USENIX Windows NT Symposium, please contact the USENIX tutorial coordinator:

Daniel V. Klein
Email: dvk@usenix.org
Phone: 412.421.0285
Fax: 412.421.2332

Usage Abstracts
All symposium participants without an accepted paper will be requested to submit a one-page abstract or summary via the symposium Web site at the time they register describing what they are doing or considering doing with Windows NT. These abstracts will be made available on the symposium Web site before the symposium and will also be distributed in paper form to attendees. The abstracts are intended to facilitate communication among attendees, helping people find others with similar interests or problems.

Program and Registration Materials
Materials containing all details of the symposium program, registration fees and forms, and hotel information will be available by April, 1999. If you wish to receive materials in print, please contact:

USENIX Conference Office
22672 Lambert Street, Suite 613
Lake Forest, CA 92630
(949) 588-8649
Fax: (949) 588-9706
Email: conference@usenix.org
URL: http://www.usenix.org

Questions
If you have questions about the symposium that are not addressed by the symposium Web site: http://www.usenix.org/events/usenix-nt99/, send mail to usenix-nt-questions@usenix.org.
Announcement and Call for Papers

2nd Large Installation System Administration of Windows NT Conference (LISA NT)

July 14-16, 1999
Westin Hotel
Seattle, Washington

Sponsored by USENIX, the Advanced Computing Systems Association
Co-sponsored by SAGE, the System Administrators Guild
Co-located with the 3rd USENIX Windows NT Symposium, August 12-14, 1999

Important Dates
Submission proposals due: Feb. 23, 1999
Notification to Authors: March 23, 1999
Final papers due: June 1, 1999

Overview
What are the qualities of good models of system and network administration? Sites around the world are asking this question as they build networks of varying size and complexity that include Microsoft Windows NT on the desktop, in the server room, or both. The Large Installation System Administration of Windows NT conference, LISA NT, is a forum to bring system administration professionals together to discuss workable solutions to the issues of administering and scaling the NT environment.

LISA NT ’99 will bring together peers and experts in our field. We invite you to submit technical papers as well as proposals for invited talks, panel sessions, tutorials, and Work-in-Progress reports. There are also opportunities for Birds-of-a-Feather sessions and demos of products and solutions.

Please review this call for papers, make a submission, and join us in making LISA NT ’99 the premiere conference for system administrators of distributed NT environments.

We look forward to your participation. If you have questions regarding submissions, acceptable topics, etc., you may e-mail us at lisantchairs@usenix.org. You may also obtain detailed guidelines for submissions by sending e-mail to lisantauthors@usenix.org.

Conference Organizers
Conference Co-chairs
Gerald Carrer, Auburn University
Ralph Loura, DE Shaw & Co.

Program Committee
Ian Alderman, Cornell University
Jeremy Allison, SGI
Phil Cox, Networking Technology Solutions
Alan Epps, Tektronix
Aileen Frisch, Exponential Consulting
John Holmwood, NOVA Gas Company
Matthew Olguin, Sematech
Andrew Rieger, Lehman Brothers
Martin Sjoelin, UBS

Topics
LISA NT is about creating a community of system administrators from diverse computing environments and giving them the opportunity to discuss problems and share solutions. Submissions may address administration methodologies (i.e., “This is what I do and why”) as well as implemented solutions (for example, “This is the tool I wrote in Perl or VBScript or any other language” or “This is how I administer X numbers of machines using tools Y and Z”). To facilitate these discussions, we encourage you to consider these topics:

Management
Management of homogeneous Windows NT networks
Management of NT in heterogeneous environments
Large-scale or distributed NT management solutions and experiences

Locally developed administration tools and techniques
Models for centralized, remote or distributed management
Reduction of the total cost of ownership for users and / or workstations
Backups and restores
User accounts

Sharing and Integration
Integration of NT into complex, existing environments
Integration of NT into UNIX environments
Integration of UNIX into NT environments
Use of cross-platform distributed services
Remote access laptops and PDAs

Tools, experiences, and issues
Successful use of third-party applications during daily administration
Surveys of examined tools (good and bad) while implementing a solution
Performance tuning and measurement
E-mail management
Central name service and browsing
Software licensing, deployment, and upgrade management
Security issues, including policies, education, response, and fixes
The registry

Best Paper Awards
Awards will be given for the best paper and best student paper at the conference.

What to Submit
The most important aspect of LISA NT is the sharing of information. With this in
mind, the program committee seeks submissions from the Windows NT system administration community in the following formats:

**Technical White Papers**
We seek papers relating work of general interest to system administrators of Windows NT, particularly technical papers that reflect hands-on experience or describe implemented or attainable solutions.

Submissions will be judged on the quality of the written submission and whether or not the work advances the art and science of NT system administration.

A paper submission should:
- contain a short abstract
- include an outline of the paper. If you have a completed paper, you may submit it instead of the abstract and outline.
- conform to the "How and Where" instructions below

You may send email to lisautauthors@usenix.org to receive detailed author instructions and a sample abstract.

Authors of an accepted paper must provide a final paper for publication in the conference proceedings. At least one author of each accepted paper will present the paper during the technical track of the conference. These presentations generally include a 20-minute talk with five to ten minutes of questions from the audience. We also ask that, if possible, copies of presentation slides be made available for publication.

Conference proceedings containing all white papers will be distributed to attendees and, following the conference, will be available online to USENIX members and for purchase from USENIX.

Note that USENIX conferences, like most conferences and journals, require that papers not be submitted simultaneously to more than one conference or publication and that submitted papers not be previously or subsequently published elsewhere. Papers accompanied by so-called "non-disclosure agreement" forms are not acceptable and will be returned to the author(s) unread. All submissions are held in the highest confidentiality prior to publication in the Proceedings, both as a matter of policy and in accord with the U.S. Copyright Act of 1976 (Title 17, U.S. Code, Section 102).

**Invited Talks/Panel Session**
If you have a presentation that does not fit in the area of a technical paper submission, please submit a proposal for an invited talk or a panel session. The proposal should
- include an extended outline of the talk or panel topic and format
- include a description of your qualifications to present the topic
- if the proposal is a panel session, list the likely participants
- conform to the "How and Where" instructions below

Invited talks generally consist of a ninety-minute talk with a short question and answer session at the end. Panels are composed of four to five people with the discussion centered on a specific topic such as security or user management. Acceptance will be based upon the general applicability of the topic and on availability of time in the program.

**Works-in-Program Reports (WIPs)**
Works-in-Program Reports (WIPs) are short talks that introduce new or ongoing work. If you have work you would like to share or an interesting idea that is not quite ready to be published, a WIP is for you. Acceptance will be based upon the applicability and scalability of the proposed solution. To submit a WIP:
- include a description of the problem and your (possibly incomplete) solution
- if necessary, include an explanation of why the problem you are addressing is interesting
- conform to the "How and Where" instructions below

**Tutorials**
On July 14, there will be full- and half-day tutorials in all areas and levels of expertise for Windows NT system administrators. Previous tutorial sessions have covered topics such as "Windows NT Security", "Windows NT Internals", "Configuring Samba, Avoiding Common Pitfalls," and "Administering Windows NT DHCP and DNS servers."

If you are interested in presenting a tutorial at LISA NT, please contact the USENIX tutorial coordinator:

Daniel V. Klein
Email: dvk@usenix.org
Phone: 412.422.0285
Fax: 412-421-2332

**Birds-of-a-Feather (BOF) and Demonstration Sessions**
BOF sessions are very informal gatherings of attendees interested in a particular topic. Demonstrations of NT tools, techniques, and products, whether freely available or commercial, are also welcome, though you will have to supply your own equipment. BOFs and demonstrations will be held on Wednesday and Thursday evenings, July 14 and 15, and may be scheduled at the conference or in advance by contacting the USENIX Conference Office.

Email: conference@usenix.org
Phone: 714.588.8649

**How and Where to Send Submissions**
Please email your submission to lisautpapers@usenix.org in any one of the following formats. If you enclose files as an attachment to your submission, please use MIME encoding, plain text with no extra markup, Postscript formatted for 8.5" x 11" page, or Microsoft Word 97 RTF or HTML. A cover letter with the following required information must be included with all submissions:

**Authors**: Names and affiliation of all authors, and an indication of which, if any, are full-time students

**Contact**: Contact for the submission
**Address**: Contact's full postal address
**Phone**: Contact's telephone number
**Fax**: Contact's fax number
**Email**: Contact's e-mail address
**URL**: For all speaker/authors if available

**Category**: Category of the submission (paper, invited talk, panel, WIP, BOF, demo/poster session)
**Title**: Title of the submission
**Needs**: Audio-visual requirements for presentation

We will acknowledge receipt of a submission by email within one week.
we're having a Blast!

We’re building desktop applications that run on UNIX, Windows 95/NT and Macintosh platforms with a native look-and-feel. We’re building Web sites with client-side applets, server-side scripting, and CGI scripts. We’re interoperating with Java, Corba, and XML. And we’re doing all of this in a fraction of the usual time, without paying royalties or licensing fees.

We're using Tcl/Tk

You’d be surprised how many companies use Tcl/Tk. Of course, many of them don’t advertise it—they consider it a strategic advantage. Read their success stories on www.scriptics.com.

It’s easy to get up and running with Tcl/Tk. Just purchase our Tcl Blast! CD-ROM. It contains Tcl/Tk and a number of different add-on packages, easy to install, ready to use, for Windows 95/NT, Macintosh, and 9 different UNIX platforms. It also contains commercial demos and documentation, along with source code for over 400 more add-on packages!

Purchase online:
http://www.linuxcentral.com
http://www.scriptics.com

JOIN THE PARTY

Convince your company to become a sponsor of the Tcl/Tk Consortium. We’re a non-profit organization helping to spread the Tcl faith. As a sponsor, you’ll receive a free subscription to our Tcl Blast! CD-ROM, discounts on Tcl/Tk products and services, and special access to our Web site. Best of all, your sponsorship will give us the means to continue our work. Visit our Web site for more details.

http://www.tclconsortium.org
CONTRIBUTIONS SOLICITED
You are encouraged to contribute articles, book reviews, and announcements to \texttt{login:}. Send them via email to <login@usenix.org> or through the postal system to the Association office.

Send SAGE material to <tmd@usenix.org>. The Association reserves the right to edit submitted material. Any reproduction of this magazine in its entirety or in part requires the permission of the Association and the author(s).

ADVERTISING ACCEPTED
\texttt{login:} offers an exceptional opportunity to reach 9,000 leading technical professionals worldwide.

Contact: Linda Barnett
barnett@usenix.org
510 528 8649