A Brief Technical Overview of the TAU Tools *

Kurt Windisch, Bernd Mohr, Al Malony
Dept. of Computer Science
University of Oregon, Eugene, OR 97403-1202

1 Introduction

TAU (Tuning and Analysis Utilities) is a visual programming and performance analysis environment based on language-level objects. Originally implemented for pC++, the elements of the TAU graphical interface represent objects of the pC++ programming paradigm: collections, classes, methods, and functions. However TAU is extensible to other programming languages, and work is currently in progress to extend TAU to C++ and HPC++. TAU is implemented in C and C++, using Tcl/Tk for the graphical user interfaces.

The TAU tools are implemented as graphical *hypertools*, each implementing some well-defined tasks. While they are distinct tools, they act in concert as if they were a single application, communicating to take advantage of each other's features. To interface with the compiler, TAU relies on language-dependent parser/unparser toolkits for instrumenting and accessing properties of program objects. TAU is also integrated with the runtime system for profiling and tracing support. The pC++ implementation of TAU uses the Sage++ [BBG+94] parser toolkit and the Tulip runtime system [BBG+93].

2 Design Overview

Figure 1 shows an detailed overview of TAU's design, consisting of the following primary components:

- The **TAU Daemon**: Serves as a central coordinator and repository for global information for all the individual TAU tools. It consists of two subcomponents: (1) a **project manager**, which manages low-level information about the project (e.g. files, paths, information about the host machine, etc) and serves as a central coordinator, relaying most inter-tool communication, and (2) a **browser database**, which indexes high-level information about the program's objects, obtained from the compiler, in order to help resolve file cross-references generated by the browser tools. The daemon is implemented in Tcl/Tk. See Figure 2.
- The individual **TAU tools**: These are the graphical tools providing a visual interface for users interaction with the program. They fall into several broad categories, each with their own requirements and consideration: static analysis tools, dynamic analysis tools, and auxilary tools. The individual tools are currently all implemented in Tcl/Tk; however, other implementation languages could be used with accomodations for inter-tool communication. They are briefly described in section 3. See Figure 2.
- The **Parser Toolkit and Compiler**: The compilation process consists of parsing the program, creating a program description file (PDF) in a language-dependant intermediate format (sometimes called a *DEP file* for historical reasons), inserting dynamic instrumentation using the parser toolkit, and generating object code from the compiler back-end. The Sage++ toolkit is used for TAU's pC++ compiler implementation. Compiler and parser components are written in C and C++.

*This research is supported by ARPA under Rome Labs contract AF 30602-92-C-0135 and Fort Huachuca contract ARMY DABT63-94-C-0029, and in part by the National Science Foundation under grant NSF ASC92-13500, by a grant from OACIS, and by IBM R&D contract MHVU3704.

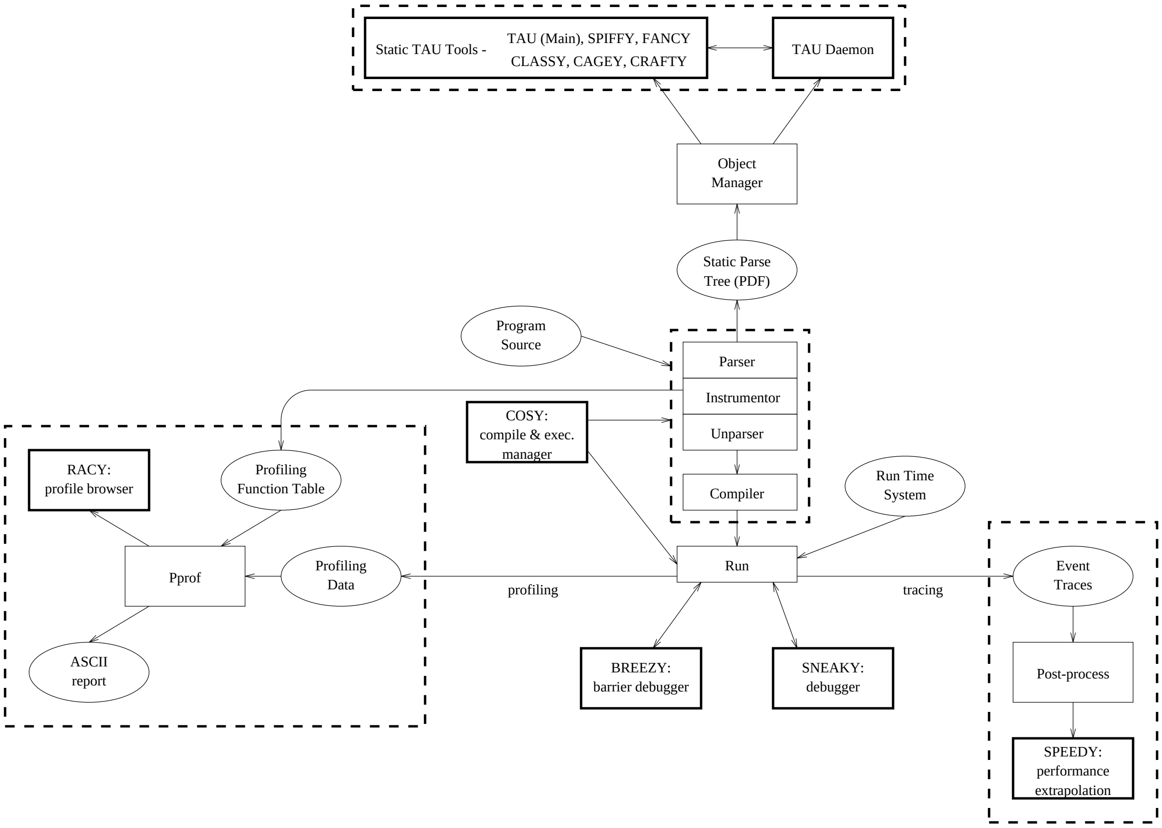

Figure 1: The architecture of the entire TAU environment.

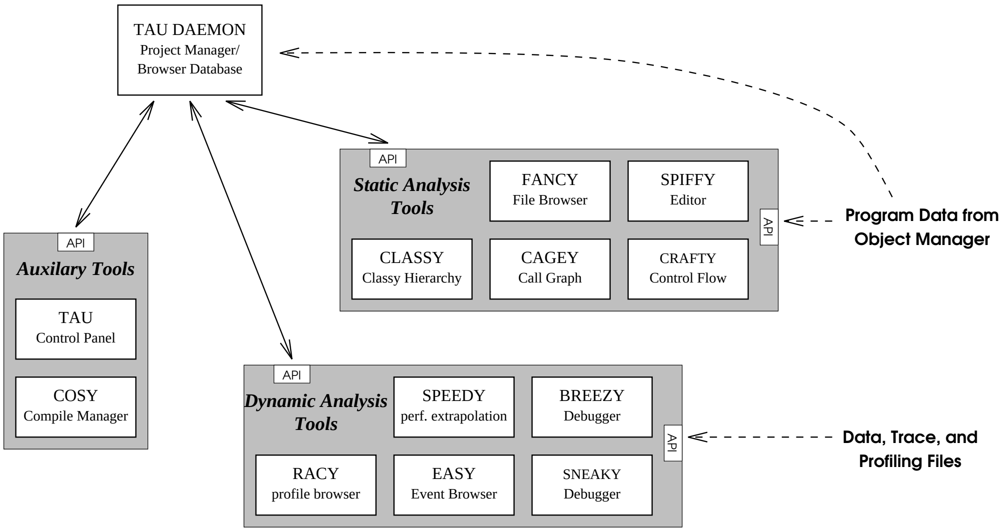

Figure 2: Internal architecture of the TAU Tools.

- The **Object Manager**: The tools access information in the PDF through the language-independant functional interface of the Object Manager (sometime called the *CGM interface*, again for historical reasons), one of which is implemented for each supported language using the language's parser toolkit. The Object Manager for pC++ is written using Sage++ in C++.
- Other Tool Support Components: Some tools, such as Racy and Speedy, require additional modules to efficiently filter profiling and tracing data. Tools are free to invoke external programs for assistance.

Because the tools are distinct from each other in design, it is relatively easy to extend TAU by creating new tools and providing them with any additional compiler or runtime support they might need.

3 Tool Descriptions

The following tools have been developed for TAU:

- **TAU** (TAU Main Control Window): provides a launching point for the rest of the TAU tools, a GUI for the project manager, and on-line, hyper-linked help.
- **COSY** (COmpile manager Status displaY): provides an interface for the compilation and execution of programs. Parameters and options for the compilation process (e.g., to compile for tracing) and for the program run (e.g., activated event classes for trace recording) may be chosen. Cosy automatically connects, if necessary, to the remote machine, executes the appropriate commands, and displays the resulting output.
- **FANCY** (File ANd Class displaY): lets you browse through the files, function, classes, and methods used in the source text of the application using a hypertext-like interface.
- **SPIFFY** (Structured Programming Interface and Fancy File displaY): a full-featured editor, allowing you to edit and browse your source files using a subset of the hypertext features found in FANCY.
- **CAGEY** (CAll Graph Extended displaY): lets you browse through the static callgraph of the application.
- **CLASSY** (CLASS hierarchY browser): displays the class hierarchy defined in the current user application.
- **RACY** (Routine and data ACcess profile displaY): a parallel profile data viewer. After compiling an application for profiling and running it, racy lets you browse through the function and collection profile data generated.
- **SPEEDY** (Speedup and Parallel Execution Extrapolation DisplaY): a graphical interface to the pC++ simulation environment XtraP. It allows performance analysis and extrapolation of pC++ programs based on execution traces.

Also under development are BREEZY and SNEAKY, breakpoint debuggers, and CRAFTY, a control-flow browser allowing for the graphical specification of instrumentation points for profiling and tracing.

4 Tool Interactions

Because of the distributed nature of TAU's tools, the interactions, both between the individual tools and with the compiler and runtime systems, are critical design elements.

4.1 Tool Internal Interactions

The individual TAU tools must interact with each other to obtain and update global project information, to invoke specific features of a given tool, or to invoke *global features*. Global features implement selection or information display operations that are common to several or all of the tools, with each tool implementing the operation using its own special capabilities. The tools are provided with a functional API encompassing these inter-tool interactions and all inter-tool communication is implemented using the remote procedure call capability of the Tk `send` command. TAU includes a Tcl/Tk proxy client which is used to forward messages to non-Tcl/Tk tools via TCP/IP sockets carrying the commands as text strings. There are plans to reimplement the TAU daemon in C as a Tcl/Tk code extension, which could incorporate this socket capability directly.

Global updates of project manager or broswer database information from individual tools are accomplished by calling API functions which inform the central TAU Daemon (project manger) of the change. The daemon then forwards the update message to each of the tools affected by the update.

Global features are implemented in a similiar way. When a tool initializes itself, it registers a list of global features for which it would like to participate in with the daemon (project manager). Then, when any tool invokes a global feature, using a provided API function, the message is sent to the Project manager, which forwards the feature invocation to all tools that have registered as recipients for that particular feature. In this way, it is trivial for TAU to be extended with any new global features required by new tools.

Finally, if a tool requires a specific feature from a single other tool, it is free to invoke that feature by remote procedure call directly to that tool, reducing bottlenecking in the Daemon.

4.2 Tool External Interactions

TAU tools must also communicate with external TAU components in order to obtain static information from the parser or dynamic profiling or tracing information from the running program.

To obtain static browsing information about the program objects, tools invoke the Object Manager executable through a Tcl/Tk API, with a flag indicating what information is needed from a parser-generated intermediate file. For example the -dumpcg flag indicates that the tool needs callgraph information relevant to the given program source code file. The Object Manager, using the parser toolkit, reads intermediate PDF files generated by the parser and extracts a complete database of the information requested by the tool, which could include function definition information, callgraph hierarchies, class hierarchies, etc. Information from the Object Manager is returned via its `stdout` in a well-defined format.

Dynamic runtime information is obtained by TAU tools by either of two distinct methods. Tools requiring profiling or tracing information can direct the compile manager tool (COSY) to intrument the program at compile time with additional code that writes information to data files. These data files can then be read and processed by the tools themselves, or by more efficient intermediate support filters. In the cases of tools requiring direct interaction with the running program, such as the BREEZY breakpoint debugger, a client is attached to the program at runtime, which communicates with the program via sockets to provide execution control and data access ability. The modular design of BREEZY allows for the sockets to be replaced by other communication protocols such as CORBA or IDL. The BREEZY infrastruction is discussed at length in [BMM95]. Furthermore, we have developed SNEEZY, a new infrastructure for dynamic runtime interaction, which will be the basis for future dynamic tools, such as the SNEAKY debugger [Han96].

5 Further Information

Refer to [MMC96] for a much more detailed look at the TAU environment. Most of the technical papers listed below are available from the TAU pages on the World Wide Web (WWW):

`http://www.cs.uoregon.edu/research/paracomp/proj/tau/papers.html`

References

[BB95] H. Beilner and F. Bause, editors. *Speedy: An Integrated Performance Extrapolation Tool for pC++ Programs*, number 977 in Lecture Notes in Computer Science. Springer–Verlag, September 1995.

[BBG+93] F. Bodin, P. Beckman, D. Gannon, S. Yang, K. Shanmugam, A. Malony, and B. Mohr. Implementing a Parallel C++ Runtime System for Scalable Parallel Systems. In *Proceedings of Supercomputing 1993*, pages 588–597, November 1993.

[BBG+94] F. Bodin, P. Beckman, D. Gannon, J. Gotwals, S. Narayana, S. Srinivas, and B. Winnicka. Sage++: An Object Oriented Toolkit and Class Library for Building Fortran and C++ Restructuring Tools. In M. Chapman and A. Vermeulen, editors, *Proceedings of the Second Annual Object-Oriented Numerics Conference*, Corvallis, OR, 1994. Rogue Wave Software.

[BHMM94] D. Brown, S. Hackstadt, A. Malony, and B. Mohr. Program Analysis Environments for Parallel Language Systems: The TAU Environment. In J.J. Dongarra and B. Tourancheau, editors, *Proceedings of the 2nd Workshop on Environments and Tools for Parallel Scientific Computing*, pages 162–171. SIAM, May 1994.

[BMM95] D. Brown, A. Malony, and B. Mohr. Language-based Parallel Program Interaction: the Breezy Approach. In *Proceedings of the International Conference on High Performance Computing (HiPC'95)*, New Delhi, India, December 1995. IEEE Computer Society, Tata McGraw–Hill.

[Han96] Lars Hansen. The sneezy manifesto. http://www.cs.uoregon.edu/research/paracomp/proj/tau/sneezy/, 1996.

[MBM94] B. Mohr, D. Brown, and A. Malony. TAU: A Portable Parallel Program Analysis Environment for pC++. In B. Buchberger and J. Volkert, editors, *Proceedings of CONPAR 94–VAPP VI*, volume 854 of *Lecture Notes in Computer Science*, pages 29–40. Springer-Verlag, September 1994.

[MMB+94] A. Malony, B. Mohr, P. Beckman, D. Gannon, S. Yang, and F. Bodin. Performance Analysis of pC++: A Portable Data-Parallel Programming System for Scalable Parallel Computers. In H.J. Siegel, editor, *Proc. Eighth International Parallel Processing Symposium*. IEEE Computer Society Press, April 1994.

[MMBG94] A. Malony, B. Mohr, P. Beckman, and D. Gannon. Program Analysis and Tuning Tools for a Parallel Object Oriented Language: An Experiment with the TAU System. In *Proceedings of the Workshop on Parallel Scientific Computing*, October 1994.

[MMC96] Bernd Mohr, Allen D. Malony, and Janice E. Cuny. Tau. In G Wilson, editor, *Parallel Programming using C++*. M.I.T. Press, 1996. http://www.cs.uoregon.edu/research/paracomp/proj/tau/papers.html.

[Moh92] B. Mohr. Standardization of Event Traces Considered Harmful or Is an Implementation of Object-Independent Event Trace Monitoring and Analysis Systems Possible? In J.J. Dongarra and B. Tourancheau, editors, *Proceedings of the CNRS-NSF Workshop on Environments and Tools For Parallel Scientific Computing*, volume 6 of *Advances in Parallel Computing*, pages 103–124. Elsevier, September 1992.

[SM95] K. Shanmugam and A. Malony. Performance Extrapolation of Parallel Programs. In C. Polychronopoulos, editor, *Proceedings of the 1995 International Conference on Parallel Processing (ICPP)*, volume II Software, pages 117–120. CRC Press, August 1995.

[SMM95] K. Shanmugam, A. Malony, and B. Mohr. Performance Extrapolation of Parallel Programs. Technical Report CIS-TR-95-14, University of Oregon, Department of Computer and Information Science, May 1995.