

Software Visualization

Within information visualization, multimedia such as images and text are used to present information in a meaningful way to a user. With the continuing trend of faster CPUs, greater memory capacity, and larger screen real estate, it is becoming increasingly possible to provide more powerful visualizations than ever before. I am particularly interested in using visualization to facilitate understanding of software systems. Such understanding is useful both to programmers who wish to debug their own programs, as well as students who want to see how an algorithm or application works.

I am interested in visualization from a systems point of view. I have focused most of my work on providing visualization in graphical debugging systems. Doing so has a number of challenges in addition to those found in algorithm animation systems. First, debuggers cannot rely on using user-modified source code containing special animation instructions, as they often work without source code altogether. Second, debuggers are notoriously machine-dependent and particular attention is required to assure portability. Finally, debuggers work with real-world applications that contain a large number of objects, so scalability is of key importance.

These challenges have been the focus on my dissertation research. I have built several fully functional systems to demonstrate the principal ideas of my research. Building systems is important, as doing so requires working out the details of the research and provides the opportunity to get feedback from users and ideas for improvement. In particular, I have implemented a graphical language (called *Tksh*), a machine-independent debugger on top of the language (called *Deet*), and two visualization systems (*Chava* and *Travis*). *Tksh* is currently used in a variety of software projects at AT&T Research. *Deet* is publicly available and has already received several thousand downloads, and has been adapted for use in embedded systems. *Chava* is currently used to teach Java in an introductory course at Drexel University.

Future Work

My long-term goal is to ease the development and understanding of software systems. This involves the creation of better user interface and human-computer interaction, as well creating infrastructure to cope with new paradigms for software development (such as the web and hand-held computing devices). I am interested in focusing on both the interface and systems issues that arise.

I plan to continue working on visualization from a systems perspective. In particular, areas such as:

1. Collaborative software development on the web and the debugging of distributed applications.
2. Application of methods used in database visualization to software visualization. Database visualization systems typically allow users to construct visual queries to display large databases with a more abstract representation. By modeling program state as a database, such tools might be useful in visualizing software.
3. Automatic detection of software patterns to simplify debugging. For example, we could perform analysis on a program to determine that is using a particular kind of data structure (such as a hash table) and automatically draw the data in a more useful way.
4. Evaluation of interfaces. In addition to publicly releasing software and gathering user feedback, I would like to set up carefully controlled user experiments to measure the effectiveness of user interfaces and visualization approaches.

Past and present research projects

My work to date has consisted of the following contributions:

TRAVIS

Algorithm animation systems and graphical debuggers perform the task of translating program state into visual representations. While algorithm animations typically rely on user augmented source code to produce visualizations,

debuggers make use of symbolic information in the target program. As a result, visualizations produced by debuggers often lack important semantic content, making them inferior to algorithm animation systems. This research introduces a method to provide higher-level, more informative visualizations in a debugger using a technique called *traversal-based visualization*. The debugger traverses a data structure using a set of user-supplied patterns to identify parts of the data structure to be drawn a similar way. A declarative language is used to specify the patterns and the actions to take when the patterns are encountered. Alternatively, the user can construct traversal specifications through a graphical user interface to the declarative language. Furthermore, the debugger supports modification of data. Changes made to the on-screen representation are reflected in the underlying data.

DEET

Deet (*Desktop Error Elimination Tool*) is a simple but powerful debugger for C and Java. It differs from conventional debuggers in that it is machine-independent, graphical, programmable, distributed, extensible, and small. Low-level operations are performed by communicating with a “nub,” which is a small set of machine-dependent functions that are embedded in the target program at compile-time, or are implemented on top of existing debuggers. Deet has a set of commands that communicate with the target's nub. The target and deet communicate by passing messages through a pipe or socket, so they can be on different machines. deet is implemented in Tksh, an extension of the KornShell that provides the graphical facilities of Tcl/Tk. Users can browse source files, set breakpoints, watch variables, and examine data structures by pointing and clicking. Additional facilities, like conditional breakpoints, can be written in either Tcl or the shell. Most debuggers are large and complicated, deet is less than 3,000 lines of code plus a few hundred lines of machine-specific nub code. It is thus easy to understand, modify, and extend. We have implemented nubs for Java and C. Deet is publicly available, and to date there have been several thousand downloads of the software.

CHAVA

Java applets have been used increasingly on web sites to perform client-side processing and provide dynamic content. While many web site analysis tools are available, their focus has been on static HTML content and most ignore applet code completely. Chava is a system that analyzes and tracks changes in Java applets. The tool extracts information from applet code about classes, methods, fields and their relationships into a relational database. A suite of programs queries the database to display structural information about the application. Other tools built on top of the database perform advanced tasks such as reachability and clustering analysis, and can graphically display query and analysis results. Databases contain supplementary checksum information which is used detect changes in two versions of a Java applet. Our tool is able to generate a database using only the compiled class files, making it possible to analyze remote applets whose source code is unavailable. Chava is publicly available for download from AT&T Labs Research.

TKSH

Tksh is a graphical language (similar to Visual Basic or Tcl/Tk) that uses KornShell (ksh93) for scripting and Tk for graphical user interface. Tksh is implemented as a ksh93 extension, and allows Tcl libraries such as Tk to run on top of ksh93 unchanged. ksh93 is well suited for graphical scripting because it is backward compatible with sh, making it both easy to learn and easy to extend existing scripts to provide user interface. Tksh also allows Tcl scripts to run without modification, making it possible to mix and match components written in either Tcl or ksh93.

COMPILER INFRASTRUCTURE

The National Compiler Infrastructure project aims to develop a common compiler platform to support the collaboration of compiler researchers and to facilitate the transfer of technology. I have worked on tools for the abstract syntax definition language (ASDL). Specifically, I have authored a graphical browser-editor of ASDL structures. The browser-editor is able to display structures as hierarchical lists or as graphical trees. By allowing the selection of colors, fonts, etc., the user can specify how each kind of node is drawn. Trees can be edited using standard cut and paste operations or by creating/modifying nodes.