

Computation Scrapbooks of Emacs Lisp Runtime State

Richard Potter

Japan Science and Technology Corporation

potter@is.s.u-tokyo.ac.jp

Abstract

Snapshots of general purpose computation states have found important but limited special purpose uses (e.g. UNIX cores and Smalltalk images). A prototype system is presented for exploring additional uses for computation snapshots in the writing, debugging, testing and documenting of computer programs. These uses center on a Computation Scrapbook, which allows complete copies of program runtime states to be easily saved, organized, and restored. Emphasis is given for using multiple snapshots to benefit intermediate level programmers who have yet to acquire the skills to be comfortable with the intermediate, changing, and mostly invisible nature of computation state. The current system allows snapshots of Emacs Lisp runtime states to be used as context for code documentation, initial and goal states for test cases, and examples for general purpose programming by demonstration.

1. Introduction

One of the many reasons why learning advanced programming skill is difficult for end-user programmers is that a program's runtime state is constantly changing yet mostly invisible. As programming skill increases, it becomes important for users to understand more of this hidden state. For example, once users create or modify code that is large enough to require layers of abstractions, the user must understand parameters and local variables, which are repeatedly and invisibly created, changed, and deleted.

It is hard to imagine one all-encompassing solution that makes any runtime state easy to understand because runtime state can be large and complex (even for small programs) and have dynamic pointers that change relationships between the changing values. Visualization techniques will surely be an important part of the most useful solutions, but it must be more than passive visualization of preset views. Also important is to give users opportunities for conceptually simple interaction with both understood and yet-to-be-understood aspects of computation state in ways that increase high-level

understanding. For end-user programming, such solutions should also be immediately useful for the current programming task at hand.

The goal of this research is to explore solutions that save and restore 100% complete snapshots of runtime state. Snapshots are only one part of the solutions, but even alone they show a bit of potential. First, saving and restoring snapshots allows user interaction with partially understood runtime state in a way that is as conceptually simple as photocopying the information on a sheet of paper. Second, snapshots are highly likely to capture aspects of the programming process that the user could benefit from understanding better. In addition, saving and restoring snapshots is relatively easy technically, and becoming more so with cheaper storage and the increased use of easily copied virtual machines. The basic idea has been used in a limited (and unglamorous) way with UNIX core dumps and in other limited ways with Smalltalk's images. Saving computation state for making objects persistent has also been investigated[1]. Additional uses are possible.

SBDebug is a prototype *Computation Scrapbook* system for saving multiple copies of the runtime state of Lisp programs from the Emacs text editor. SBDebug provides the infrastructure for exploring specific solutions that increase opportunities to interact with partially understood runtime state in productive ways while reading, writing, debugging, testing, and documenting programs. The system architecture and user interface lessons from this system are being used to plan a scrapbook system for the Java virtual machine.

2. The Users

The intended users have programming aptitude, but do not have the time to master general purpose programming skills or the patience to endure prolonged setbacks. They could benefit from advanced skills because they have reached the limits of an easier-to-use programming technique such as spreadsheets, programming by demonstration, or a domain-specific technique. These people have reached a "wall" or "cliff" in the learning curve. In other words, they long for what has been called a Gentle-Slope System[2]. For these users, it is probably

impossible to eliminate the need for programming skill, and so it is important to provide effective ways to acquire it, ideally in an incremental yet continuously useful way. Nevertheless, the primary goal of the users is the actual use of the programs they create.

3. Three Solutions using Snapshots

One simple use for a computation scrapbook is to support detailed source code documentation. A user could be studying example code and be curious why certain lines are necessary. Comments on those lines could have links to snapshots that illustrate exactly what the lines do on a specific example. The programmer who created the link to the snapshot could insure that the example is simple enough to understand but still be interesting, perhaps illustrating important special cases.

Looking at specific examples can give users fresh viewpoints that help them confirm or reevaluate high-level understanding of the code. They are encouraged to form hypotheses because they can more easily test them. Specific examples can also help users keep moving productively when stuck or confused at the source code level. A computation scrapbook facility can increase opportunities for this interaction because users do not have to re-create illustrative runtime states by starting the program and setting debugger breakpoints. Clicking a link to a snapshot is much easier and can even take users to difficult-to-re-create states involving partially completed loops or non-deterministic events.

Probably the most useful way that multiple snapshots can be used is to provide initial conditions and goal states that contain enough information to test arbitrary segments of code. A learning user can then test a few lines of code to see if it handles a particular special case. The user does not have to wait until the whole module is finished and then write a separate driver routine to test the code. Testing early and often can make the code better and build the user's confidence. The increased interaction with runtime state can increase the user's understanding.

Notice how this use of computation snapshots is synergistic with the previous use. If the pairs of snapshots from test cases are retained and linked to the source code, they can document the specific examples that the user considered when writing the program.

A challenging but interesting use for multiple snapshots is as examples for a programming by demonstration (PBD) system. If the user can edit runtime states, then the editing operations and snapshots of the intermediate states can be used as input to a programming by demonstration system that generates general purpose program segments. Like the use above for test cases, this PBD snapshot use can potentially work for arbitrary segments of code. Therefore it can be applied to small

(but potentially tricky) code segments where PBD techniques can be tractable.

For users trying to understand runtime state, this gives more opportunities to work with concrete examples in a productive way. Users who began programming using PBD techniques will appreciate the opportunities to continue using PBD, even though the program state is more complicated and might not have a task-oriented graphical user interface as in previous PBD research. Even if the PBD system is unable to infer the correct program, the increased interaction with runtime state can lead to high-level insights. At the very least, the edited snapshots can serve as test cases for code that the users are struggling to write.

4. The Emacs Prototype

All three of the above solutions can be demonstrated using the current version of SBDebug, a Computation Scrapbook system for Emacs Lisp. SBDebug is an extension of Emacs's *edebug* source debugger. It includes functionality to save and restore computation state and a scripting system to implement the above solutions. A simple visualization of stack bindings and a backwards stepping mechanism have also been added.

The advantages of using Emacs for this first prototype are the flexibility of Lisp for fast prototyping and the stability of Emacs that will allow the system to be distributed unchanged to a sizable user community. The *edebug* debugger has been particularly easy to extend. The snapshots are only approximations, but adequate for the initial phases of the research. Current work is to advance it from a demoable to a distributable system.

5. Conclusions

Although the idea of 100% complete computation snapshots is conceptually simple, there are many technical challenges to address. For example, using snapshots for test cases requires that old snapshots be merged with new code. As such challenges are addressed, the goal is to keep the user view of the process as conceptually close to the original idea as possible. Despite the challenges, it already seems clear that Computation Scrapbooks expose enough low-lying fruit that even the most-straightforward solutions will lead to practical benefits.

[1] M. Jordan and M. Atkinson, "Orthogonal Persistence for the Java™ Platform: Specification and Rationale", Technical Report TR-2000-94, Sun Microsystems Laboratories, M/S 29-01, 901 San Antonio Road, Palo Alto, CA 94303, USA, December 2000.

[2] M. Dertouzos, "Creating the People's Computer", *Technology Review*, MIT, Cambridge, MA, April 1997, pp. 20-28.