Paul V. Gestwicki
pvg@cse.buffalo.edu

Interactive Visualization of Object-Oriented Languages

Committee:

Dr. Bharat Jayaraman, Chair
Dr. Ashim Garg
Dr. Bina Ramamurthy
Dr. Li Lin (Industrial Engineering)

Abstract

We present a novel approach for the interactive visualization of the execution of object-oriented programs that fulfills several important criteria: (i) visual depiction of the current execution state as well as the entire history of execution; (ii) support for forward as well as reverse execution stepping; (iii) enhanced graph-drawing techniques for the current state and history of execution; (iv) support for run-time queries on the execution state; (v) support for all major features of the Java language and use of the standard Java compiler and run-time system. Taken together, our approach marks a significant advance over existing techniques for visualizing object-oriented program execution.

Our proposed notation for expressing Java execution states clarifies the important fact that objects are environments; that is, a method execution is shown within its object context. By incorporating diagrams similar to those used during the design phase, we help to close the loop between design and execution. In particular, an enhanced UML-like object diagram is used for depicting the current state, and sequence diagrams are used for execution history. Interactive execution is instrumented by conceptualizing program execution as a database, so that stepping backward or forward through recorded states involves rolling back or re-committing transactions. Our methodology incorporates a novel technique for recording minimal state transitions in order to efficiently provide this interactive execution.

Automatically generating object and sequence diagrams requires advanced graph drawing techniques. The enhanced object diagrams we present are not simple graphs, and special techniques are required to process them. We illustrate how layered drawing techniques can be used to achieve good drawings of enhanced object diagrams. Two techniques are presented for automatically drawing sequence diagrams: a stochastic simulated annealing approach and a fast greedy technique. We show how traditional graph-theoretic approaches can be combined with program-specific properties in order to produce better automatic drawings. Specifically, an analysis of the class diagram is used to determine the classes that form logical clusters, and these clusters are formed in the object diagram generated at runtime. We have implemented our approach in a prototypical tool called Jive: Java Interactive Visualization Environment.