Query-based Debugging and Visualization in JIVE

Jeffrey K. Czyz
Department of Computer Science and Engineering
University at Buffalo
The State University of New York

Adviser: Dr. Bharat Jayaraman
Motivation

– “Software bugs, or errors, are so prevalent and so detrimental that they cost the U.S. economy an estimated $59.5 billion annually.”

National Institute of Standards and Technology
http://www.nist.gov/public_affairs/releases/n02-10.htm

– According to the same study, $22.2 billion can be saved through more effective identification and removal of bugs.
Motivation

– Debugger technology has not changed much
  • Setting breakpoints
  • Spying on variables
  • Stepping forward in execution
  • Examining variables using the call stack

– Modern debuggers are still mainly **procedural and textual** in nature
Declarative and Visual Approach

– Visualize execution history and state to clarify object interactions and object structure
  • Enhanced Object Diagram (execution state)
  • Sequence Diagram (execution history)

– Provide an extensible set of queries over a program’s execution history and individual runtime states
  • Formulate queries using diagrams or source code
  • Report results as diagram annotations

– Allow revisiting previous runtime states
Declarative debugging complements procedural debugging just as web searching complements web browsing.
What is JIVE?

– Interactive Execution Environment for Eclipse
  • Declarative and Visual Approach to Debugging
  • Pedagogic Tool
– Extension of the JDT Debugger
  • Interactive Visualization
  • Query-based Debugging
  • Reverse Stepping
Principles of JIVE

- Support Full Java Language
- Depict Objects as Environments
- Visualize Current State and Execution History
- Provide Multiple Views of Runtime State
- Produce Aesthetic Layouts
- Support Forward and Reverse Execution
- Support Declarative Queries on Runtime States
- Use Existing Java Technology
JIVE: Java Interactive Visualization Environment

JIVE Perspective

Current State

Execution History
Interactive Execution

– JIVE depicts the current runtime state and execution history of a program in a visual manner
  • Enhanced Object Diagram
  • Sequence Diagram

– Diagrams are formed dynamically at runtime
  • Object Diagram changes as methods are called and returned, objects are created, variables assigned, etc.
  • Sequence Diagram grows as the program executes

– Each point on the Sequence Diagram corresponds to a runtime state (depicted by the Object Diagram)
Object Diagram

- Depicts the current runtime state
- Method activations shown within their proper object contexts
- Call path shown
- Clarifies object structure
- Expand/Collapse objects
- Show/Hide member tables
Sequence Diagram

- Depicts history of execution
- Method activations shown along object lifelines
- Expand/Collapse method activations
- Jump back to previous runtime states
- View search results
- Clarifies object interactions
Query-based Debugging

- Search individual states or execution history
  - Object Attribute and Local Variable Changes
  - Class/Object Invariant Checking
  - Object Creations
  - Method Activations
  - Statement Executions
  - Exceptions Thrown/Caught

- Result reporting
  - Diagram annotations
  - Tabular form
  - Call path to event
Reverse Stepping

- Often bugs are not discovered until after the errant statement executes
- JIVE supports stepping forward or backward through execution to review past states
- Can jump immediately back to a point of interest using the diagram or query results
Future Work

– Reduce Overheads of Data Collection
– Produce Scalable Visualizations
– Runtime Flow Analysis for ‘Why’ Queries
– Database Support for Long Executions
– Efficient Jumping to Previous States
– Closer Integration with the JDT Debugger
JIVE Web Site

– http://www.cse.buffalo.edu/jive/
– Download using update manager
– Subscribe to RSS feed for latest news and releases
– Comments and suggestions welcome

Thank you!