SomeTitle

Tørresen, Håvard

Supervisor:
Trætteberg, Hallvard

March 4, 2014

Abstract

Background:

Results:

Conclusion:

Acknowledgements

Contents

List of Figures

List of Listings

1 introduction

2 Task Description

3 Prestudy

There are several ways a debugger can aid the programmer beyond just showing the current state of a program. For a fresh programmer, either in general, or at a certain project, the most useful method is probably to generate diagrams that visualize the current state, and the path of execution. I.e. some form of object-, and/or sequence–diagram. Such diagrams can make it easier to get an overview of a programs current state, and to understand how it works. In order to generate the diagrams, the tools can analyze both the source-code, and an execution trace, depending on the type of diagrams to generate.

In order to discover the source of a problem, especially for large programs, it may be useful to enable the programmer to execute queries concerning the current, or previous program-states. By querying previous states, one can find the reason why certain variables have their current value, and discover the real source of a problem, instead of the spot where a fatal error occurs.

Methods:

Visualization:
Generating graphs and diagrams representing the program
Easier to get an overview of program structure and execution
generation based on code itself, or trace of program execution, former easiest to use for class diagrams, latter for sequence diagrams and other types of runtime representation.

Interactive forwards- and backwards-stepping
two forms: re-execution, state-saving
re-execution: small memory footprint, slow backward stepping
state-save: fast stepping both ways, needs more memory, amount depending on program. Slower in general due to overhead of saving every change of program-state, but can be fast enough to not be noticed. Will again depend on the program.

Queries:
fast way to check object-relations and -properties
Ask the debugger to evaluate a statement concerning the state of the debugged program. E.g. checking constraints or invariants, making sure relations are correct, why did something happen, etc.

Tools:

GNU debugger (GDB)
tracing, reverse debugging, general debug-stuff
multiplatform, multi-language
remote debugging
CLI-only, needs separate front-end

Jinsight
made by IBM
two components: profiler and visualizer
only for z/OS or Linux on system z
builds a trace when application is running
client connects to profiler and visualizes the trace
modified JVM?
120 minute trace limit
very powerful

Javavis
relies on the Java Debug Interface (JDI), and the Vivaldi Kernel (a visualization library)
shows dynamic behavior of running program
object diagrams+sequence diagram, UML
smooth transitions
not a debugger

code canvas (visual studio)
unites all project-files on a infinite zoomable surface
both content and info
layers of visualization - files/folders, diagrams, tests, editors, traces ++
several layers visible at the same time
search

trace viewer plugin (g-Eclipse)
g-eclipse=grid, archived project
visualize and analyze communication of message-passing programs - communication graphs
standalone/platform independent
designed for massive parallelism - MPI and similar
debugging
events are marked by different colored nodes in the graphs.

Whyline
Interrogative debugger
why did, why did not
works on recorded executions

TOD: Trace-Oriented Debugger
omniscient debugger
queries
dynamic visualizations - high-level, graph of event density

Jive
combines all fields
contour diagram - Enhanced object diagram, showing objects and their environments: fields, values, relations, inheritance, etc.
sequence diagram - generated during execution, supports zooming and folding to cope with, and hide irrelevant information, but can still become quite large.

stepping - state-saving enables fast backward stepping, and the current state is reflected in the diagrams.
queries - enabled by state-saving. Allows filtering of irrelevant information.
can be used for debugging

4 Conclusion

References