SomeTitle

Tørresen, Håvard

Supervisor:
Trætteberg, Hallvard

March 12, 2014

Abstract

Background:

Results:

Conclusion:

Acknowledgements

Contents

List of Figures

List of Listings

1 introduction

New students may find programming in general, and object oriented programming specifically, difficult. Can be hard to understand concepts, and get a mental model of what is going on. Especially code generated by GUI-builder tools. Traditional debuggers are not necessarily helping when detecting a run-time error. Tools that present the state of a program in a simple visual way may help to understand.

2 Task Description

3 Prestudy

3.1 methods

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.

Execution traces can also be used to enable backwards stepping of program execution. Stepping back in time allows the user to not only see the failure state of a program, but to go back and see what caused the problem, instead of adding a new breakpoint and running the program again.Each reverse step can be faily cheap, but it may still be impractical to make large jumps in the execution history.

One can avoid the potential disadvantages of manual backstepping by using queries instead. Queries enable the user to asks the debugger about the current and earlier states of execution in a simple way. The debugger then does the work of finding what was asked for, instead of the user manually searching through the program states.

3.2 Existing Tools

There currently exists several tools that provide one or two of the methods mentioned above. GDB offers a tracing environment, but due to its command-line interface it is not necessarily easy to use on its own. The Trace Viewer Plugin for g-eclipse, uses a trace to generate visualizations of the program execution, and thus makes it easier to understand, but is designed for massive parallelism, and may not be very useful for understanding smaller programs. Whyline, and Trace-Oriented Debugger also utilize execution traces, but use them to enable querying, instead of providing visualizations. JAVAVIS provides visualizations in the form of UML-diagrams, but does not provide any debugging features. Code Canvas uses an interesting way of visualizing an entire project, everything from source-code to design documents and diagrams are layered onto a large canvas, allowing easy navigation between various elements, but is restricted to Microsoft Visual Studio. Jinsight is a powerful tool built by IBM, supporting both tracing and visualization. However, it is restricted to z/OS and linux on system Z, preventing most people from using it.

Jive seems to be the only tool that utilizes all three methods, as well as being freely available as a plugin for eclipse, making it easy to install and use. During program executiion, Jive generates a contour diagram, and a sequence diagram.Combined with an execution trace, it allows the user to jump back and forth in the execution, and have the diagrams updated accordingly.

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

Glossary

Code Canvas TODO. 4

Execution trace A log of all changes to the state of a program throughout its execution. 4

GDB GNU debugger. A multiplatform, multilanguage CLI-debugger with tracing. 4

JAVAVIS Tool to visualize running Java applications. 4

Jinsight TODO. 4

Jive TODO. 4

Trace Viewer Plugin todo. 4

Trace-Oriented Debugger TODO. 4

Whyline TODO. 4

References