

SomeTitle

Tørresen, Håvard

Supervisor:  
Trætteberg, Hallvard

March 10, 2014

## Abstract

**Background:**

**Results:**

**Conclusion:**

## Acknowledgements

## Contents

1	introduction	2
2	Task Description	3
3	Prestudy	4
4	Conclusion	6
	Glossary	7
	Bibliography	7

## 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.

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.

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 - communi-

cation 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

**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

## References