Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.
BEAT: A VISUALIZING DEBUGGER FOR CONCURRENT, SHARED MEMORY JAVA PROGRAMS.

A thesis presented in fulfillment of the requirements for the degree of
Master of Science
in
Computer Science

At Massey University, Palmerston North, New Zealand.

PAUL THOMAS JOHNSON
February 2011
Abstract

This thesis presents our research into the creation of a new concurrency visualization called Beat. Our research began with the observation that software used to create and record music has been incredibly successful and that there were broad similarities between these pieces of software and existing concurrent visualizations. This led us to question if there were reasons why music software had been successful, why concurrent visualizations hadn’t been as comparatively successful and finally if this could teach us anything about building better visualizations.

The existing literature was examined to learn more about concurrency, visualization and music software and notations. For concurrency we wanted to see what existing solutions to the problems of concurrency exist and in particular existing solutions to concurrent debugging. With existing visualizations we wanted to see if there was anything we could identify about them that made them ineffective for solving the problems of concurrency. Conversely we wanted to see if we could identify what made music software and notations so effective for solving problems for musicians.

The examination of the existing literature led to the design and implementation of the Beat software, based on the ideas we had discovered in our search of the literature. To test how effective the design and software was we conducted an evaluation with programmers, which validated many of the decisions we had made and gave us a number of future directions to take with our research.

The main contribution of this thesis is a new approach to designing concurrency visualizations that emphasizes using the low level details of program execution and integrates execution data into a single view to help with the error debugging process. This in contrast to existing concurrency visualizations that focus on the debugging of performance problems.
Acknowledgements

Without the tireless support, encouragement, supervision and regular prodding of Dr Stephen Marsland this thesis would never have been completed. Thanks Stephen.

Thanks to Dr Elizabeth Kemp for valuable advice regarding testing the Beat software.

Thanks to Professor Hans Guesgen for help planning testing of the software.

Approval for the evaluation was granted under the low risk guidelines provided by the Massey University Human Ethics Committee.
# Contents

1 Introduction 1
   1.1 Motivation ................................................. 1
   1.2 Thesis Overview ............................................ 2

2 Literature Review 4
   2.1 Introduction ................................................ 4
   2.2 Introduction to Concurrency ................................. 4
      2.2.1 Instruction Level Parallelism .......................... 5
      2.2.2 Data Parallelism ....................................... 7
      2.2.3 Task Concurrency ...................................... 7
      2.2.4 Models of Process Communication ....................... 8
      2.2.5 Concurrency and the Operating System ................ 12
      2.2.6 Debugging Concurrent Programs ....................... 15
   2.3 Introduction to Visualization ............................... 20
      2.3.1 Elements of Visualization ............................. 21
      2.3.2 Visualization Data .................................... 24
   2.4 Software Visualization ...................................... 27
      2.4.1 Visual VM .............................................. 28
      2.4.2 PARADE ............................................... 33
      2.4.3 Understanding Complex Multithreaded Software Systems by Using Trace Visualization ............................ 37
      2.4.4 Zinsight ............................................... 38
      2.4.5 TIE: Thread Interleaving Explorer ..................... 42
      2.4.6 UML Sequence Diagrams ................................. 43
      2.4.7 Visual Programming Languages ......................... 48
      2.4.8 Common Features Summary ................................ 51
   2.5 Temporal Notations from outside Software Visualization .......... 52
CONTENTS

2.5.1 Music Notation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
2.5.2 Non-Linear Audio Editing . . . . . . . . . . . . . . . . . . . . . . . 54
2.5.3 Comparison to Software Visualization . . . . . . . . . . . . . . . . 58

3 Design 60
3.1 Design of Beat . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
3.2 Visualization Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
3.3 Space . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
3.4 Text . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
3.5 Lines, Colors and Icons . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
3.6 Loops, Branches, Errors and Exceptions . . . . . . . . . . . . . . . . . 66
3.7 Context Switches . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
3.8 State . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
3.9 Concurrency Errors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70
3.10 Interaction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72

4 Implementation 74
4.1 Implementation Choices . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
4.1.1 Language . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
4.1.2 Standalone program versus IDE . . . . . . . . . . . . . . . . . . . . 75
4.1.3 Gathering Trace Data . . . . . . . . . . . . . . . . . . . . . . . . . . 76
4.1.4 Visualization display choices . . . . . . . . . . . . . . . . . . . . . 82
4.2 Plugin Infrastructure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 82
4.2.1 Launcher and the JDT . . . . . . . . . . . . . . . . . . . . . . . . . . 83
4.3 Data Gathering . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 84
4.3.1 Trace data annotations . . . . . . . . . . . . . . . . . . . . . . . . . . 84
4.3.2 Source Code instrumentation . . . . . . . . . . . . . . . . . . . . . . 85
4.3.3 Runtime method probes . . . . . . . . . . . . . . . . . . . . . . . . . 87
4.3.4 Data loading and processing . . . . . . . . . . . . . . . . . . . . . . . 87
4.3.5 Context Switches and DTrace . . . . . . . . . . . . . . . . . . . . . 90
4.3.6 Program State and Concurrency errors . . . . . . . . . . . . . . . . 91
4.4 Visualization Generation . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91
4.4.1 Ruby and Templating . . . . . . . . . . . . . . . . . . . . . . . . . . . 91
4.4.2 Text Positioning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 92
4.4.3 Canvas Object and Threads . . . . . . . . . . . . . . . . . . . . . . . . 96
4.4.4 JQuery and CSS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 97
## Contents

### 5 Evaluation

5.1 Evaluation Introduction .................................................. 98  
5.2 Evaluation Method ....................................................... 98  
  5.2.1 Tasks and Timing ..................................................... 99  
  5.2.2 Previous Experience .................................................. 99  
  5.2.3 Timing and Experience Questions ................................. 100  
  5.2.4 UI Questions and Comments ......................................... 100  
5.3 Timing and Quantitative Questions Results .......................... 101  
  5.3.1 Was using Beat significantly faster? ............................. 101  
  5.3.2 Was Beat helpful in one not the other? .......................... 102  
  5.3.3 Was using Beat significantly easier? ............................. 102  
  5.3.4 What effect did experience have? ............................... 102  
  5.3.5 Did the order that Beat was used in have any effect? .......... 103  
  5.3.6 Was one of the tasks harder than the other? .................. 103  
5.4 Visualization Design Questionnaire Results ......................... 104  
5.5 Comments From Questionnaires ........................................... 104  
  5.5.1 Like Comments ....................................................... 105  
  5.5.2 Minor UI problems .................................................. 105  
  5.5.3 Major Dislikes/ Needed Features .................................. 106  
5.6 Observations of Participants ............................................. 106  
5.7 Evaluation Limitations ................................................... 107  

### 6 Conclusion

6.1 Conclusion ................................................................. 108  
6.2 Future Work .............................................................. 110  
  6.2.1 Testing ............................................................... 110  
  6.2.2 Core Improvements .................................................. 110  
  6.2.3 UI improvements ................................................... 111  
  6.2.4 Debugger and Static Analysis ................................... 112  
  6.2.5 Education ........................................................... 112  
  6.2.6 Visual Language .................................................... 112  

### A Questionnaire

113

### B Evaluation Source Code

B.1 Source Code for Deadlock Task ........................................... 114  
B.2 Source Code for Race Condition Task ................................. 114
C Result Tables

C.1 Timing and Experience Results ................................. 124
C.2 UI Questions ....................................................... 125
C.3 Comments ............................................................ 125
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Pipelined instruction execution</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Race condition diagram</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Race condition fixed with locks</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>Two processes in a deadlock. The processes have each acquired a lock</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>but are both waiting on the lock held by the other. Since the processes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>are unable to acquire the locks they need to continue they are unable to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>progress.</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td>Processes messaging shared account process</td>
<td>11</td>
</tr>
<tr>
<td>2.6</td>
<td>Screenshot of Intellitrace [14] showing a list of the events in an executing</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>program the system has traced, including exceptions and user interface events.</td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>Treemap example [9] that shows the amount of downloads for software</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>projects on sourceforge. This is an example of using space to convey</td>
<td></td>
</tr>
<tr>
<td></td>
<td>amount and the relationship of the amount of one thing to another.</td>
<td></td>
</tr>
<tr>
<td>2.8</td>
<td>Examples for continuous data from Tory and Miller</td>
<td>26</td>
</tr>
<tr>
<td>2.9</td>
<td>Examples for discrete data from Tory and Miller</td>
<td>26</td>
</tr>
<tr>
<td>2.10</td>
<td>Process overview for VisualVM [24] showing some basic bits of information</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>about a process including the execution arguments and the process id.</td>
<td></td>
</tr>
<tr>
<td>2.11</td>
<td>Monitor view of VisualVM [24] that shows various graphs showing different aspects of a</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>programs performance. The top left shows the CPU</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and Garbage collection activity. The top right shows the memory usage</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and the type of memory being used. The bottom left shows information</td>
<td></td>
</tr>
<tr>
<td></td>
<td>about classes and the bottom right shows information about threads.</td>
<td></td>
</tr>
<tr>
<td>2.12</td>
<td>Thread timeline view of VisualVM [24]. The timeline shows the thread</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>state over time.</td>
<td></td>
</tr>
<tr>
<td>2.13</td>
<td>Sampler view of VisualVM [24].</td>
<td>32</td>
</tr>
</tbody>
</table>
2.14 Profiler view of VisualVM [24] showing how long methods are taking to execute. ................................................................. 33
2.15 Threads view of Parade [54, 66] showing a list of threads as boxes. Colour is used to show state. ................................. 34
2.16 Functions view of Parade [54, 66] shows a trace of what functions have executed for a single thread. ................................. 35
2.17 History view of Parade [54, 66] showing the timeline of thread execution, somewhat like the Visual VM example. ................................. 36
2.18 Mutex view of Parade [54, 66]. The small coloured circles are threads that move inside the larger clear circle that represent the Mutex when they acquire it. ................................................................. 36
2.19 Barrier view of PARADE [54, 66] showing what threads have entered and exited a barrier synchronization point. ................................. 37
2.20 Trace Visualization showing two threads [62]. Each thread “lane” has an overview (top) and zoom in panel on the bottom that show what functions have been executing. ................................................................. 38
2.21 Event view of Zinsight [33]. Note that time flows vertically rather than horizontally like the previous examples. ................................. 39
2.22 Statistics view of Zinsight [33]. Some what like the profiling view of Visual VM this provides statistical information about the running program. ................................. 40
2.23 Sequence Context view of Zinsight [33]. This is showing a summarization of the various execution paths the program has gone through. ................................. 41
2.24 Thread Interleaving Explorer software [42]. Note the numerous different panels used to provide enable a user to zoom in from the more general information at the top to more specific information in the bottom columns. ................................. 42
2.25 UML collaboration diagram [23]. The numbers show the order of method calls between the object instances which are represented by the boxes. ................................. 43
2.26 Basic UML sequence diagram [12]. Time flows vertically with the lines representing the execution history of the object instances and the narrow blocks on the lines showing when a method was called on an object. ................................. 44
2.27 UML Sequence diagram with combined fragments [12]. The boxes are used to represent various aspects that are not covered by the lines and execution boxes. ................................................................. 45
2.28 Diagram of saUML [65]. Colour is used to represent thread state and the rounded corner boxes show monitor state information. ................................. 46
2.29 Context switches in Artho’s UML extension [27]. Note the hexagon threads and the dashed lines that show when a thread starts executing in the object instances. .................................................. 47
2.30 Condition variable wait and notification in Artho’s UML extension. Note the dotted lines which demonstrate notification of a lock to the waiting thread. .................................................. 47
2.31 Overview of JIVE [36] showing the various panels. The top left shows the object instances and there values the bottom left shows a sequence diagram of the execution. ........................................ 48
2.32 Multiple threads in JIVE [36]. Colour is used to differentiate the threads. 49
2.33 LabVIEW – an example of a ”boxes and wires” visual language [20][6]. ”boxes” are operations on a stream of data that are carried between the operations by the ”wires”. Some ”boxes” will be used for input and output. 50
2.34 Basic music notation ................................................. 53
2.35 Different note lengths ............................................... 53
2.36 Text and lines indicating volume. .............................. 53
2.37 Speed notation for music notation ............................... 54
2.38 Glyphs used in music notation .................................... 54
2.39 Track view of Logic Audio ......................................... 55
2.40 Piano roll view from Logic Studio .............................. 56
2.41 Sound edit view from Logic Studio ............................. 57
3.1 Use of space in Beat ................................................. 62
3.2 Method calls to object instances represented as blocks in the object tracks. 62
3.3 Design with source code included in method call boxes. .... 63
3.4 Adding lines showing path of execution between method blocks. .... 64
3.5 Thread state being shown by icons and dashed lines. ........ 65
3.6 Thread looping shown by repeated loop body. ................. 66
3.7 Branches shown by only showing the content of the branch taken. 67
3.8 Exceptions in a program thread shown by T, C and E icons. .... 68
3.9 Context switch in Beat ............................................... 69
3.10 State changes shown by additional lines ....................... 70
3.11 State only mode for method blocks. Source code is hidden and only changes to program variables are shown. ................. 71
3.12 Possible method of highlighting a program deadlock. ........ 72
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Class diagram of Beat model relationships</td>
<td>89</td>
</tr>
<tr>
<td>4.2</td>
<td>Space compression in method blocks</td>
<td>94</td>
</tr>
<tr>
<td>4.3</td>
<td>Parallel thread execution</td>
<td>95</td>
</tr>
</tbody>
</table>
# List of Tables

5.1 Participants and tasks performed ................................. 99
5.2 Comparison of time taken on both tasks .......................... 101
5.3 Correlations between Eclipse and concurrency experience ........ 103
5.4 Comparison of Beat order performance ............................ 103
5.5 Task difficulty comparison ........................................ 104
5.6 Participants rating of what was task was easiest .............. 104
## List of listings

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Example for Instruction Parallelism</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Simple example of a future in Java [5], the line containing future.get() is where the</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>other thread is started and the main thread blocks waiting for the value to be generated.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Probing an open() system call with DTrace [22]. The lines containing “syscall” are the</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>events in the kernel to trace and the line beneath is a predicate which checks whether</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the action contained in the body of the trace will execute. The trace body is written in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a programming language allowing complicated tracing actions to be performed.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>This example shows how to launch a program using the JDI library [15].</td>
<td>77</td>
</tr>
<tr>
<td>5</td>
<td>Registering to receive method entry events from remote program [15].</td>
<td>77</td>
</tr>
<tr>
<td>6</td>
<td>Method to receive registered events from the debugger [15].</td>
<td>78</td>
</tr>
<tr>
<td>7</td>
<td>Probing methods using AspectJ pointcuts.</td>
<td>80</td>
</tr>
<tr>
<td>8</td>
<td>Example of part of an ANTLR grammar used for processing a block of Java code to add</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>tracing probes.</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Configuration of a launcher extension for Eclipse in the XML plugin configuration.</td>
<td>83</td>
</tr>
<tr>
<td>10</td>
<td>Diagram showing basic tree structure of Java source code. A classes is made up of methods</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>which contain blocks of statements.</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Code to insert method probes around a method call. The lines containing</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>ExpressionStatement are the code for the methods to insert. The lines like x.add(beforeExp)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>add the method probes.</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Diagram showing the result of inserting probe methods before and after method call.</td>
<td>89</td>
</tr>
<tr>
<td>13</td>
<td>Probing scheduling in the kernel</td>
<td>91</td>
</tr>
<tr>
<td>14</td>
<td>ERB templating code</td>
<td>93</td>
</tr>
<tr>
<td>15</td>
<td>Javascript line drawing code</td>
<td>96</td>
</tr>
<tr>
<td>No.</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-----</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>16</td>
<td>Contains main method of deadlock test.</td>
<td>115</td>
</tr>
<tr>
<td>17</td>
<td>Printer object for deadlock test.</td>
<td>116</td>
</tr>
<tr>
<td>18</td>
<td>First printer thread for deadlock test part 1.</td>
<td>117</td>
</tr>
<tr>
<td>19</td>
<td>First printer thread for deadlock test part 2.</td>
<td>118</td>
</tr>
<tr>
<td>20</td>
<td>Second printer thread for deadlock test part 1.</td>
<td>119</td>
</tr>
<tr>
<td>21</td>
<td>Second printer thread for deadlock test part 2.</td>
<td>120</td>
</tr>
<tr>
<td>22</td>
<td>Contains main method of race condition test.</td>
<td>121</td>
</tr>
<tr>
<td>23</td>
<td>Printer thread for race condition test.</td>
<td>122</td>
</tr>
<tr>
<td>24</td>
<td>Shared object for race condition test.</td>
<td>123</td>
</tr>
</tbody>
</table>