Beat: A tool for visualizing the execution of object orientated concurrent programs

Paul Johnson and Stephen Marsland
School of Engineering and Technology
Massey University
Palmerston North, New Zealand
p.johnson@clear.net.nz, s.r.marsland@massey.ac.nz

ABSTRACT

The transition from single core to multicore processors has lead to a greater need for programmers to become familiar with concurrent programming. In particular there is a need to help programmers new to concurrent programming to understand how to utilize multiprocessor systems. We feel that visualization tools could be extremely helpful to these programmers. In this paper we present a novel design for a program trace visualizer we call Beat.

Categories and Subject Descriptors

D.2.5 [**Testing and Debugging**]: Tracing

Keywords

Software visualization, Concurrency

1. INTRODUCTION

The design of Beat is similar to UML sequence diagrams but modified to use an approach inspired by media software such as video and audio editors. Beat is implemented as Eclipse plugin for concurrent Java programs and uses HTML to display the generated visualization. Our goals for Beat are to create a tool useful for debugging concurrent programs and also for assisting programs learning to program concurrently.

2. DESIGN

Rather than representing object instances as a "life-line" like in UML[6] we use "tracks" like video editing software[1] with the methods being executed represented as blocks that take up the entire track. Our reason for doing this is because it allows us space to include the source code text of the method that is being executed. We feel that this will help programmers using our software as a learning tool to relate their source code to its concurrent execution.

Like Uml we use lines to represent flow of control or a thread through the instances.

Similar to systems such as saUML[7] and also performance visualizations such as Suns VisualVM[5] we make extensive use of color, however we use it to differentiate the threads of a program instead of indicating thread state. To provide indications of thread state we use icons representing concurrent actions and dashed lines to represent when a thread is inactive.

Basic interactions such as resizing and reordering columns are provided.

We feel that our approach is visually less cluttered than existing solutions while still communicating a great deal of information.

One limitation of our approach is the amount of screen needed to effectively display the amount of information produced, our experiences developing and using the visualization suggest a minimum screen size of about 22" is required to be effective.

Unlike UML we don't intend that Beat will be a tool for visualizing programs in general, the use of the source code ties it to a single language, to support more than one language would require at least some redesign.

3. IMPLEMENTATION

We chose to implement Beat for the Java language using the Eclipse IDE[3].

Our implementation is divided into 4 parts: plugin infrastructure that coordinates the process of running the program and creating the visualization, a preprocessor to instrument code with information probes, a simple runtime class to gather and record the data and the visualization itself.

The plugin infrastructure coordinates tasks such as running the preprocessor, compiling the preprocessed program, running the preprocessed program, loading the data and displaying the visualization. This is implemented as a launcher extension for Eclipse which allows users to launch the visualization using the green run arrow used to run regular Java programs.

Much of the work of implementing the plugin went into creating the preprocessor to instrument the code to add methods that record events and the time they occur within a

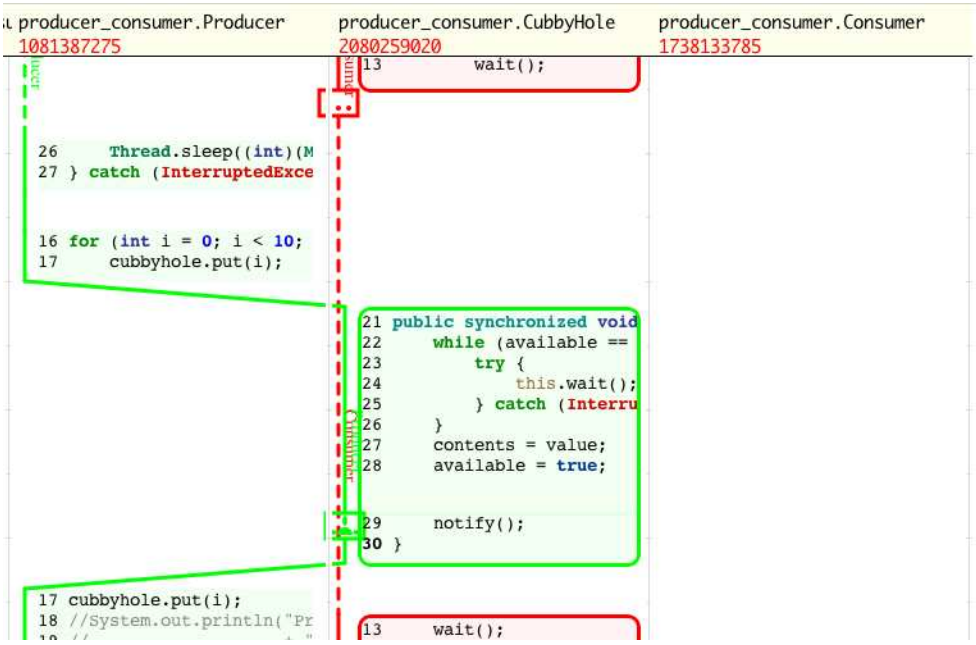

Figure 1: Beat Screenshot

program. To do this we used part of the Java Development Tools (JDT)[2] which come with Eclipse. The JDT contains a framework for parsing and creating an abstract syntax tree that can be manipulated and written back out to a file to add the probes.

The second part of the data gathering is a simple class which contains the probe method which adds the recorded event to an in memory list, when the class receives a thread or program exit message it writes the in memory data out to a file named after the thread that is being traced.

Finally we apply a number of fixes to the raw data to make it suitable for display. HTML5[4] is used to display the visualization as it handles text well and is something we have a great deal of experience with. An HTML template file is used to generate the text portions and is styled and positioned using CSS. The thread lines and icons are drawn using the canvas vector drawing object that is part of the HTML 5 standard. A web browser embedded in Eclipse is used to display the generated file.

The obvious limitation to our implementation is that it suffers from the "probe effect" where observing the program changes it, unfortunately we haven't made any measurements of this effect all though we have tried to limit its effect by carefully programming. Our software is available for download at http://github.com/pj/beat by using the git version control system.

4. CONCLUSIONS

Though there are design and technical limitations to our approach a number that Beat supports our goal of making a tool useful for debugging and for programmers learning concurrency. By implementing our software as an Eclipse plugin for Java we make it easy for people to use and integrate with their existing programming setup.

5. REFERENCES

[1] Apple - final cut studio - final cut pro 7, 2010.
[2] Eclipse java development tools (jdt) overview, 2010.
[3] Eclipse.org home, 2010.
[4] Html5 (including next generation additions still in development), 2010.
[5] visualvm: Home, 2010.
[6] J. Arlow and I. Neustadt. *Uml and the Unified Process: Practical Object-Oriented Analysis and Design.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
[7] S. Xie, E. Kraemer, R. E. K. Stirewalt, L. K. Dillon, and S. D. Fleming. Assessing the benefits of synchronization-adorned sequence diagrams: two controlled experiments. In *SoftVis '08: Proceedings of the 4th ACM symposium on Software visualization*, pages 9–18, New York, NY, USA, 2008. ACM.