Using Dynamic Analysis to Create Trace-Focused User Interfaces for IDEs

Del Myers
University of Victoria
Victoria, British Columbia
delmyers.cs@gmail.com

Margaret-Anne Storey
University of Victoria
Victoria, British Columbia
mstorey@uvic.ca

ABSTRACT

This research demonstration presents the tool, *Dynamic Interactive Views for Reverse Engineering* (Diver). Diver supports software understanding through a *trace focused user interface*. The trace focused user interface is a method of re-organizing the user interface of integrated development environments so that developers can focus their attention on artifacts related to the run-time behaviour of the software that they are investigating. The tool combines concepts from research in software visualization, dynamic analysis, software reconnaissance, and task focused user interfaces.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments

General Terms

Human Factors

Keywords

Reverse Engineering, Integrated Development Environments, Software Reconnaissance, Visualization

1. INTRODUCTION

The complexity of current software systems makes them difficult to understand. Modern IDEs offer extensive support through advanced features such as searching for program elements, free-form text queries, and visualizations of type hierarchies. However, this support is limited by language features such as polymorphism and dynamic binding. Dynamic analysis may be employed to overcome these limitations. This method of analysis typically logs runtime information of in a *dynamic execution trace*. Traces can be analyzed to extract information about a running system and visualized using UML sequence diagrams.

In our previous work [1], we conducted a study to determine the requirements for supporting reverse engineering tasks using sequence diagrams within an integrated development environment (IDE). We found that engineers require linked views of both the static and dynamic aspects of their program to effectively understand their software. It is particularly important for the views to be integrated with source code. Our findings led us to design the *Dynamic Interactive Tools for Reverse Engineering* (Diver)[1] tool.

Diver provides a completely new perspective for a developer's IDE, through what we refer to as a *trace focused user interface* (UI). The user interactively creates a set of execution traces that are displayed in a special view, called the *Program Traces View*. Traces in this view can then be selected by the user to both filter and highlight relevant information about the program execution. The user can now interactively explore this information using standard IDE views linked to our custom sequence diagram view and trace search features. This trace focused UI approach is unique because it is the first one to combine research in software visualization [1], dynamic analysis [2], software reconnaissance [5] and task focused user interfaces [3].

2. DIVER AND THE TRACE FOCUSED USER INTERFACE

Diver is implemented as a set of plug-ins for the Eclipse integrated development environment.[2] It offers a perspective that gives the user several views which help him or her interact with source code and dynamic traces. All of the views are linked to provide a rich interaction experience for the user. Each of the views are designed or modified to allow users to gain insight about the runtime behaviour of their software through the use of dynamic execution traces. Execution traces are created by the user through the standard Eclipse launch facilities and through specialised actions contributed to the Eclipse *Debug View* (Fig. 1-A).

Central to the Diver perspective is the *Program Traces View* (Fig. 1-B) which shows execution traces created by the user. The traces are organized by launch name and the date and time of the trace. Each thread of execution contained in the trace is also accessible from this view.

The user can use the *Program Traces View* to open a visualization of a thread in Diver's custom *Sequence Diagram View* (Fig. 1-C). Diver sequence diagrams are unique in that they visualize runtime as well as source code information. This is achieved through an algorithm which matches method calls in the trace to the blocks of source code. These matchings are used to display combined fragments that contain method calls and to compact iterations of loops. Our empirical study indicates that compacting loop iterations can reduce the size of the sequence diagrams for industrial software by up to 80% [4]. The sequence diagram is also

FSE-18, November 7–11, 2010, Santa Fe, New Mexico, USA.
ACM 978-1-60558-791-2/10/11.

[1] `http://diver.sf.net`
[2] `http://www.eclipse.org`

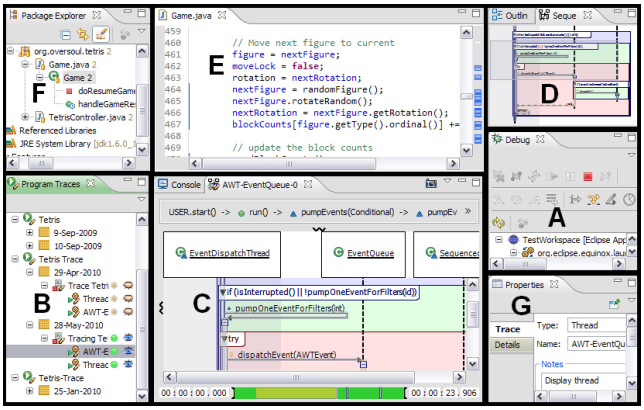

Figure 1: The Views of Diver

fully interactive. Lifelines, combined fragments, and activation boxes can all be expanded or collapsed to increase or decrease the amount of information that is viewed. Every element in the view is selectable, and the view supports keyboard navigation. The view can also be navigated using a linked outline view (Fig. 1-D).

The Program Traces view can also be used to "activate" a trace. Activating a trace does several things to change the Eclipse user interface. First, it provides a code coverage visualization. The Eclipse *Source Code Editor* (Fig. 1-E) is marked with annotations that indicate the lines of code on which methods were called during the trace.

Activating a trace also initiates software reconnaissance. Software reconnaissance is a method for locating features in source code by comparing information logged in multiple execution traces [5]. Diver filters The Eclipse *Package Explorer* (Fig. 1-F) so that only program elements that are referenced in the active trace are visible. The user is also able to use the *Program Traces View* to "hide" traces. This is done by activating a trace and selecting one or more traces to hide in the view. Hiding traces adjusts the Diver filter so that only source code elements in the active trace, but not the hidden traces, are visible. This enables users to locate features in source code by focusing on program elements that are unique to those features.

The Eclipse *Package Explorer* is also linked to the Diver *Sequence Diagram View*. When a trace is active, users can right-click on a class or method to reveal, in the *Sequence Diagram View*, the first time it was referenced in a selected thread in the trace. Other references to the selected method or class can be revealed using a timeline widget which is displayed at the very bottom of the sequence diagram. The timeline is marked by vertical bars representing all of the calls to the selected element in the *Package Explorer*.

Finally, every element in a trace may be annotated using the standard Eclipse *Properties View* (Fig. 1-G). Annotations can later be searched using the Diver Trace Search. Wild-card searches for classes and methods are also supported.

3. IMPLEMENTATION DETAILS

Diver is implemented in Java and C++ and consists of more than 35K classes. It uses the Java Virtual Machine Tooling Interface[3] to generate traces of Java applications or Eclipse Rich Client Platform (RCP) applications. Traces contain data about method and constructor calls and returns. Users can control which method and constructor calls are stored by using a button supplied in the Eclipse *Debug View*. The data is stored in binary files which are later analyzed and indexed into a database. Information indexed into the database can be controlled through customisable "trace analysis filters". By default, Diver only indexes information related to source code in the project that defines the Java application, or the plug-ins that define the Eclipse RCP application. The process is scalable in our experience. We have used Diver to analyze large software applications such as the Eclipse IDE, the Jetty HTTP Server[4] and the HSQLDB database.[5]

4. CONCLUSIONS AND FUTURE WORK

Diver combines research on trace visualization, software reconnaissance, and task focused user interfaces to create a unique user interface for program exploration. We have performed a user study in which participants were asked to use Diver to locate and analyze features of Diver itself. The results of this study are promising and have been submitted for publication. We are also currently logging information from current users of Diver to investigate how they use the tool. Diver is available as open source from our web site: `http://diver.sf.net`.

5. REFERENCES

[1] C. Bennett, D. Myers, M. Storey, D. German, D. Ouellet, M. Salois, and P. Charland. A survey and evaluation of tool features for understanding reverse-engineered sequence diagrams. *Journal of Software Maintenance and Evolution: Research and Practice*, 20(4):291–315, 2008.

[2] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke. A systematic survey of program comprehension through dynamic analysis. *IEEE Trans. on Software Engineering*, 35(5):684 –702, 2009.

[3] M. Kersten and G. C. Murphy. Mylar: a degree-of-interest model for ides. In *Proc. of the 4th Int'l. Conf. on Aspect-Oriented Software Development*, pages 159–168, New York, USA, 2005. ACM.

[4] D. Myers, M.-A. Storey, and M. Salois. Utilizing debug information to compact loops in large execution traces. In *Proc. of the 14th European Conference on Software Maintenance and Re-engineering*, pages 41–50. IEEE, March 2010.

[5] N. Wilde and M. Scully. Software reconnaissance: mapping program features to code. *Journal of Software Maintenance: Research and Practice*, 7(1):49–62, 1995.

[3] `http://download-llnw.oracle.com/javase/6/docs/technotes/guides/jvmti/index.html`
[4] `http://www.eclipse.org/jetty/`
[5] `http://www.hsqldb.org`