Effective Exploration and Visualization of Large Execution Traces

Abdelwahab Hamou-Lhadj
ECE, Concordia University
1455 Maisonneuve West, Montreal, Quebec, Canada
abdelw@ece.concordia.ca

Abstract

Understanding the behaviour of a software system can be made easier if dynamic analysis techniques are used. Run-time information is typically represented in the form of execution traces. Raw traces, however, can be extremely large - often millions of lines long. In previous work, we presented a tool called SEAT (Software Exploration and Analysis Tool), which is a trace visualization tool that supports several features for rapid exploration of lengthy traces. In this paper, we present the new features supported by SEAT, namely, the ability to plug-in new trace filtering algorithms, a usable control widget called PictureTree for displaying traces, and several new views that display useful information about the trace under study.

Keywords: Reverse engineering, dynamic analysis, design recovery, program comprehension.

1. Introduction

Understanding the behaviour of software systems is an important aspect in developing effective strategies and tools for program comprehension. The behaviour of a software system is typically represented using execution traces. Raw traces, however, tend to be overwhelmingly large, which has led to the development of many trace analysis tools (e.g. [1, 4]). In [3], we surveyed a large number of tools and found them limited in the way they handle large and most complex traces.

In previous work [2], we presented an Eclipse plug-in tool called SEAT (Software Exploration and Analysis Tool), which is designed to help software engineers manipulate traces of routine (method) calls. With SEAT, a software engineer can collapse/expand parts of the trace, detect patterns of execution, search for specific components, reduce the size of the trace by collapsing similar (but not necessarily identical) sequences of calls, etc.

We have improved SEAT by redesigning its user interface and adding new features, among which the most interesting ones consist of a new control widget called PictureTree for efficient display of the trace content, the ability to plug-in third-party trace filtering algorithms, and several supporting views that aim at providing guidance to software engineers during the exploration of the trace content.

2. SEAT

A snapshot of the new graphical user interface of SEAT is shown in Figure 1; we can see that the workbench is divided into four parts. The upper left is the navigator where the traces as well as the system from which they are generated are displayed making it possible for software engineers to explore the traces at the same time as other system's artefacts. The upper right area is the default editor area; a trace can be explored in several independent explorations using a multiple-page editor. In SEAT, traces are displayed as tree structures using an innovative tree widget called PictureTree that is described in Section 2.2. The lower left area consists of a control panel used to display properties such as statistical information of the trace under exploration and the current node. The lower right area is a set of auxiliary views used to view different entities such as the results of a search query, trace patterns, etc.

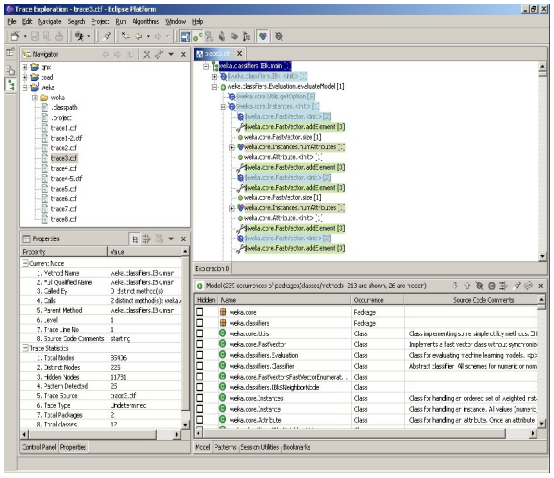

Figure 1. Snapshot of SEAT's enhanced user interface

New Trace Filtering Algorithms

SEAT supports several trace filtering techniques to help make a trace appear simpler and smaller. The most important ones are based on criteria by which various parts of a trace can be treated as the same pattern. The tool also has built-in algorithms that automatically hide various low-level implementation constructs such as accessing methods, methods of inner classes, constructors, etc.

We have improved SEAT in such a way that new trace filtering algorithms can be added easily. We achieved this by providing a well defined framework for trace filtering algorithms. The framework requires that a newly developed algorithm implement an interface called *IAlgorithm* so it can both apply and revoke the filtering algorithm. An abstract class, called *Algorithm,* that implements *IAlgorithm,* is provided to ease the integration of a new algorithm into the framework.

After the algorithm has been developed, the next step is to describe its attributes and parameters according to an Eclipse extension point called "seat.algorithm" that is provided with SEAT (the schema of this extension point is not provided here owing to limited space). The basic attributes of an algorithm include a unique identifier, name, implementation class, icon, description, etc. An optional attribute for an algorithm is parameter, with which a user can define several input parameters for a given algorithm.

PictureTree: A Usable Trace View

Traditionally a non-leaf tree node has two states: expanded or collapsed. In our trace exploration context, because various filtering algorithms can be applied to the trace, some nodes in the trace can be hidden if they meet certain criteria. As a result, in a certain exploration point, a non-leaf node can have the following expanded states: fully expanded, that is, all the nodes are shown; and partially expanded, meaning some or all the children are hidden by some filtering algorithms, so only part of children are currently displayed. Existing tree widgets either expand or collapse the entire subtree. To address this issue, we created a new tree widget called PictureTree that works as follows:

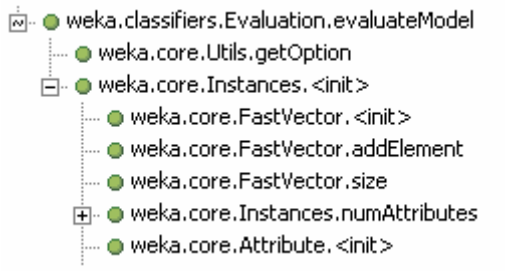

Figure 2. A Snapshot of PictureTree

When a subtree is first displayed, it is in collapsed state, indicated by the icon '+'. When a user wants to expand the subtree to visit its children nodes, the icon that represents the new state will change to either '–' icon or '~' icon depending on whether there are children nodes being hidden by any filtering algorithms. If no children nodes are marked filtered, '-' icon will be displayed for the subtree. Otherwise, '~' icon will be used to indicate that some children cannot be seen. By using Ctrl+Click, the user can switch between these two icons, and refresh children nodes respectively. Figure 2 shows an example of using PictureTree. The node labelled "evaluateModel" is partially expanded, i.e., the subtree rooted at this node contains nodes that are hidden.

Support of Multiple Exploration Views

SEAT supports several new views that can help speed up the process of exploring the content of a trace. These views are:

- The Model View: It is used to display the distinct elements invoked in the trace.
- The Search View: It provides the result of a search query. Search history is recorded automatically and the user can easily return to a previous result.
- A Bookmark View: It provides the user with the ability to record the locations in the trace previously visited and return to it when necessary.
- The Pattern View: It displays patterns of executions in a list, from which the user can select the patterns that need to be analyzed.
- The Utility View: It contains the components that are considered by software engineers as utilities.

3. Conclusions and Future Directions

In this paper, we presented the new features supported by our trace analysis tool, SEAT. The objective is to help software engineers effectively and efficiently explore the content of large execution traces.

Future directions should focus primarily on the evaluation of SEAT by experimenting with large traces. There is also a need to investigate how the proposed techniques can be applied to other types of traces such as inter-process communication.

References

[1] W. De Pauw, E. Jensen, N. Mitchell, G. Sevitsky, and J. Vlissides, J. Yang, "Visualizing the Execution of Java programs", In *Proc. of the International Seminar on Software Visualization*, LNCS 2269, Springer-Verlag, 2002, pp. 151-162.

[2] A. Hamou-Lhadj, T. Lethbridge, and L. Fu, "SEAT: A Usable Trace Analysis Tool", In *Proc. of the 13th International Workshop on Program Comprehension*, IEEE CS, 2005, pp. 157-160.

[3] A. Hamou-Lhadj and T. Lethbridge, "A Survey of Trace Exploration Tools and Techniques", In *Proc. of the 14th IBM Conference of the Centre for Advanced Studies on Collaborative Research*, IBM Press, pages 42-55, 2004.

[4] D. Jerding, and S. Rugaber, "Using Visualization for Architecture Localization and Extraction", In *Proc. of the 4th Working Conference on Reverse Engineering*, 1997, pp. 219-234.