STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

JSET - JAVA SOFTWARE EVOLUTION TRACKER

ARTHUR-JOZSEF MOLNAR

Abstract. This paper introduces the Java Software Evolution Tracker, a visualization and analysis tool that provides practitioners the means to examine the evolution of a software system from a top to bottom perspective, starting with changes in the graphical user interface all the way to source code modifications.

1. Introduction

Software tools occupy an important place in every software practitioner's toolbox. They can assist in virtually all activities undertaken during the life of software starting from requirements analysis to test case design and execution. By studying the evolution of widely used IDE's such as Eclipse [3] one can see that each new version ships with better and more complex tools for aiding professionals in building higher quality software faster. Modern environments include tools for working with UML artifacts, navigating source code and working with a wide variety of file types.

However, modern day software systems fall into many categories, each having unique requirements, artifacts and processes. Our goal is to incorporate this knowledge into tool development efforts and use the latest available results from research in developing new, useful tools for practitioners in software development.

The rest of this paper is structured as follows: the next section will present the work jSET is based on. The third section will describe the tool in detail, while the fourth overviews its current limitations. The last section is reserved for conclusions and future work planned.

Received by the editors: March 30, 2011.

2010 *Mathematics Subject Classification.* 68N15.

1998 *CR Categories and Descriptors.* D.2.2 [**Software**]: Software Engineering – *Design Tools and Techniques*.

Key words and phrases. Software visualization, Software tool, Static analysis.

2. Related work

The development of jSET was made possible by two tools that come from the academic environment. They are presented in the following paragraphs together with earlier efforts of using them for software visualisation.

The first of these tools is called GUIRipper, part of the comprehensive GUITAR toolset [2]. GUIRipper acts on a GUI driven target application [7] that it runs and records all the widgets' properties across all the application's windows. The resulting GUI model is saved in XML format for later use. When developing jSET, additional functionality for recording widget event handlers and capturing screenshots was programmed into GUIRipper. This modified version can be found at the jSET website [9].

The second application is the Soot analysis framework [8]. Soot is a static analysis framework that targets Java bytecode; all its implemented analyses are performed without running the target application. One of the most important artifacts produced by Soot is the application's call graph: a directed graph that describes the calling relations between the application's methods [4]. As it is computed statically, it does not provide information regarding order of calling or execution traces. For use with jSET, a suitable model that persists the computed callgraph was developed and implemented as a wrapper over Soot.

The applications described above laid the groundwork for the development of advanced software tools. Some of these earlier efforts, that served as inspiration for jSET's development are discussed in the following paragraphs.

Possibly the earliest of such tools is JAnalyzer [1], *a visual static analyzer for Java* developed by Bodden et al. JAnalyzer leverages call graph information generated by Soot and graphically displays the calling relations in a program. It also allows viewing the source code of compiled methods, thus creating a link between the application bytecode and its sources.

A more advanced undertaking is described in [5] where the author presents a call graph comparison tool that ranks the differences according to their importance. The same paper also introduces a browser application for navigating call graphs, similar to JAnalyzer.

3. jSET - Java Software Evolution Tracker

jSET is a software analysis and visualisation tool created for practitioners and researchers alike. The three main ideas guiding its development are:

(1) Provide an advanced visualization tool for easily accessing static analysis results inside a software project environment without the need for a laborious setup phase.

(2) Integrate the obtained results in the context of GUI driven applications by offering a seamless transition from the GUI's level all the way to the application's source code.
(3) Facilitate identification and analysis of changes across versions of a software system from changes in the GUI down to source code modifications.

In order to use jSET, the first step is to create a project. A jSET project is an XML file that contains information about the locations off all the necessary artifacts. For a valid project, the following data is required:

- The GUI model obtained by running our modified version of GUIRipper on the target application.
- The callgraph obtained by running jSET's Soot wrapper on the target application.
- The target application's bytecode (including used libraries)
- The target application's source code[1].

It is important to note that all the steps of building a project can be easily automated. Both GUIRipper and our Soot wrapper can be executed via command line and manual intervention using configuration files is required only for certain changes in the target application such as specifying special handling for some GUI elements (e.g: exempting components from analysis) or updating the application's libraries. As such jSET is easily integrateable with the target application's build system. More detailed information and project examples are available on the jSET website [9].

The jSET application can be used in two modes: project exploration and project comparison. When starting the application, the user must select one or two projects to load. Selecting one defaults jSET to project exploration mode. The comparison mode can be used to display the differeces between the target application's versions[2]. Figure 1 shows the tool in comparison mode, the target application being an early version of the open-source FreeMind mind mapping software[3].

The tool's user interface is rather similar for both modes but because of differences in the information displayed, the following paragraphs present both in detail, starting with project exploration mode.

3.1. **Project exploration mode.** jSET's user interface consists of several panes displaying information about the loaded project. The left-side pane

[1]Only if viewing the source code is desired

[2]The projects should capture the same application at different versions, however this is not enforced

[3]https://sourceforge.net/projects/freemind/

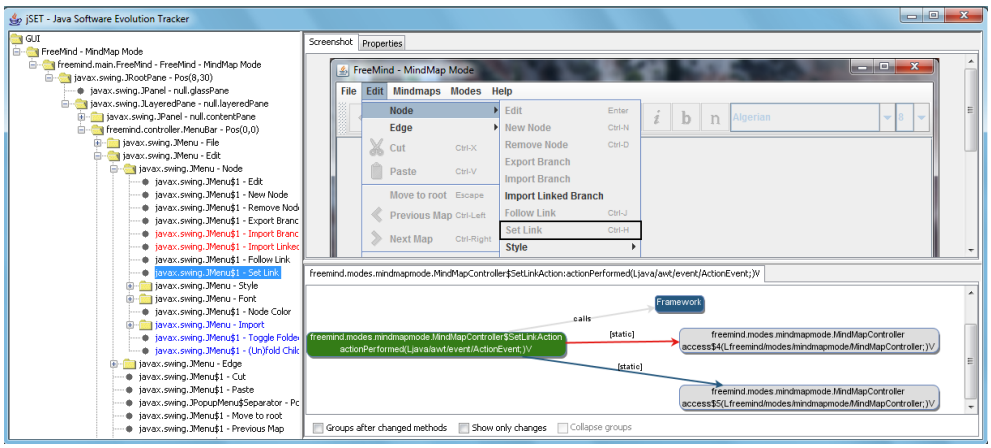

FIGURE 1. jSET in Project Comparison mode

displays the target application's user interface hierarchy. When a widget is selected from it the right hand tabbed pane displays its important properties[4] and a screenshot showing it as it appears when the target application is running, shown in Figure 1.

Among the displayed properties we can find the handlers associated for the widget's events. Some of them are attached by the platform as they control behind the scenes aspects regarding the GUI. Others however are defined by the application itself. One of jSET's original contributions concerns visualising the target application's event handling. When selecting one of the handlers, the relevant part of the application's call graph is displayed in the right lower pane. Here the user can examine what methods might be run when a certain event is fired (e.g: a button is clicked on the GUI). It is important to mention that the displayed graph is only part of the application's call graph, as the entire structure is too complex to be displayed at once for all but the most trivial programs [5].

3.2. **Project comparison mode.** Since the compare view's layout is similar to the project exploration one, this section will discuss only the relevant differences.

The first difference regards the GUI tree shown on the left hand side. While the exploration mode displays the target application's GUI hierarchy, the comparison mode also displays the differences between the projects GUIs. Looking at Figure 1, we can see certain items from the hierarchy are color coded. Red items represent widgets that can no longer be found on the newer

[4]Like swingExplorer does (http://www.swingexplorer.com)

version, while items in blue are widgets not found on the older one. Green items represent widgets affected by underlying changes in their event handler code.

The second way compare mode differs from exploration regards the graph display. For widgets identified in both loaded versions, a new way of displaying call graphs was developed. The astute reader will notice the same color coding used as in the case of GUI widgets, this time customized for method calls. As such, the displayed graph will actually be the reunion of the event handlers' call graphs across versions: red edges represent calls removed from the newer version while blue edges show new method calls. Green is used for methods that underwent code changes between the versions. Right clicking a node brings up a menu that allows comparing the bytecode or source code (if available) of the selected method between versions.

The jSET application is an ongoing effort of providing practitioners with tools based on the latest accomplishments in research. The exploration mode is useful for understanding how the target application works by identifying events that cause code to run and analyzing the calling relations between the application methods. The comparison mode allows tracing the evolution of a software system; this enables evaluating the changes across versions in a top to bottom fashion, from the GUI level to source code statement changes.

4. Limitations

Although much thought went into the design and implementation of jSET, there are some aspects that limit its usability. Some of them stem from inherent limitation of the tools jSET itself is based on. The following list attempts a brief overview of these limitations:

- *Dynamic user interfaces.* Some applications create and dispose of GUI elements dynamically which sometimes makes it impossible to save a complete model of the GUI.
- *Reflection.* Applications using reflection might instantiate classes and run methods that are not captured in the callgraph, leading to an incomplete representation in jSET.
- *Interacting widgets.* In some cases, events fired on widgets might fire new events on other GUI elements. This is not accounted for by the current version of jSET, and in these cases library callbacks might occur that are not captured in the displayed graphs.

5. Conclusions and future work

In this paper we presented jSET, a new software visualisation and analyis tool. jSET introduces a new approach for software visualization that goes from

GUI to source code statement level. The tool implements a new way of displaying changes in the target application's GUI across versions and integrates this information with changes in the source code.

Future work planned includes devising new algorithms for identifying equivalent GUI elements across program versions, improving the state of the tool's limitations and using jSET as the software visualisation tool of a fully automated regression test procedure for GUI based applications [6].

Acknowledgements

The author was supported by programs co-financed by The Sectoral Operational Programme Human Resources Development, Contract POS DRU 6/1.5/S/3 - "Doctoral studies: through science towards society"

References

[1] Eric Bodden, *Janalyzer: A visual static analyzer for java*, (2003), As contribution for the SET Awards 2003, category computing.

[2] Daniel Hackner and Atif M. Memon, *Test case generator for GUITAR*, ICSE '08: Research Demonstration Track: International Conference on Software Engineering (Washington, DC, USA), IEEE Computer Society, 2008.

[3] Daqing Hou and Yuejiao Wang, *An empirical analysis of the evolution of user-visible features in an integrated development environment*, Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research (New York, NY, USA), CASCON '09, ACM, 2009, pp. 122–135.

[4] Ondrej Lhotak, *Spark: A flexible point-to analysis framework for java*, Tech. report, McGill University, Montreal, 2002.

[5] Ondrej Lhotak, *Comparing call graphs*, Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering (New York, NY, USA), PASTE '07, ACM, 2007, pp. 37–42.

[6] Atif Memon, Adithya Nagarajan, and Qing Xie, *Automating regression testing for evolving gui software*, Journal of Software Maintenance **17** (2005), 27–64.

[7] Atif Muhammed Memon, *A comprehensive framework for testing graphical user interfaces*, Ph.D. thesis, 2001, AAI3026063.

[8] Vijay Sundaresan, *Practical techniques for virtual call resolution in java*, Tech. report, McGill University, 1999.

[9] Website., https://sourceforge.net/projects/javaset.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Department of Computer Science, 1 M. Kogalniceanu, Cluj-Napoca 400084, Romania

E-mail address: arthur@cs.ubbcluj.ro