A Case Study of Dynamic Analysis to Locate Unexpected Side Effects Inside of Frameworks

Izuru Kume, Nara Institute of Science and Technology, Ikoma, Japan
Masahide Nakamura, Kobe University, Kobe, Japan
Naoya Nitta, Konan University, Kobe, Japan
Etsuya Shibayama, The University of Tokyo, Tokyo, Japan

ABSTRACT

Recently many frameworks are deployed without proper documents to explain their correct usage. In the absence of proper documents, application developers often write code to call a framework API in a wrong way. Such a wrong API call tends to bring about a failure after its complex chain of infection inside of a framework. The complexity and the lack of implementation knowledge about a framework make it difficult for application developers to debug this kind of failure. In the preceding study the authors focused on unexpected side effects that are caused by wrong API calls and bring about failures, and developed a dynamic analysis technique to detect such side effects. In this paper, the authors introduce a case study to find a wrong API call using our technique.

Keywords: Application Framework, Debug, Dynamic Analysis, Object-Oriented Programming Language, Program Understanding, Side Effect, Software Engineering

INTRODUCTION

Recently many frameworks are used in software development without proper documents to explain their correct usage (Shull, Lanubile & Basili, 2000). As a result, application developers often write code to call APIs provided by frameworks in wrong ways (Monperrus & Mezini, 2013). Several static analysis techniques (Monperrus & Mezini, 2013; Mishne, Shoham & Yahav, 2012) are proposed to solve this problem, but they don’t cover such wrong API calls that maintainers can find them faulty only by examining their runtime conditions such as the reference structure among involved objects, the timing of inversion of controls to trigger the API calls, and etc.

DOI: 10.4018/IJSI.2015070103
We pursue a method to debug wrong API calls as defects. In general, debugging a failure requires an examination of source code and its execution. Such a task is necessary in finding a chain of infection (Zeller 2009). In this task, maintainers are required to find erroneous states based on their implementation knowledge of the system under debugging. For example, they should find that some value of a local variable is faulty, or a method is invoked at a wrong timing.

Maintainers of a framework application, who are application developers and suspect a wrong API call, try to trace back the execution from a failure to this wrong API call. Because such a wrong API call produces an erroneous state inside of the framework, they have to examine the source code of the framework and its runtime states. Usually they are new to the implementation details of the framework, and thus their task to detect a chain of infection inside of the framework is very difficult and time consuming.

To cope with this problem, we leverage possibly unexpected side effects which seem to be hidden from frameworks, and cause failures via outdated objects’ state. In our preceding study (Kume, Nitta, Nakamura & Shibayama, 2014), we developed a dynamic analysis technique to detect such hidden updates and uses of outdated states in a program execution trace. In this paper, we introduce a case study where we found wrong API calls by detecting an unexpected side effect using our technique.

The rest of this paper is as follows: In section PRELIMINARY, we introduce basic concepts of framework applications, and we also explain the difficulty to debug wrong framework API calls. In section PROPOSED APPROACH, we explain an overview of our dynamic analysis technique. In section CASE STUDY, we introduce our case study, and discuss the usefulness and limitation of our technique in section DISCUSSION. Section RELATED WORK is for explaining our related work, and we state our conclusion in CONCLUSION.

PRELIMINARY

Operations, Statements, and Dependency

We assume that frameworks and their applications are implemented in Java language. Java objects consist of class instances and arrays. Operations on objects are method invocations (except for static methods), and accesses to their instance variables or array components. An operation is expressed as a statement or an expression in a statement.

Parameters of a method invocation consist of its receiver (if any) and arguments. We call instance variables and array components persistent variables. When a persistent variable of an object is accessed to assign or get a value, then we say that the object is used as a carrier of the value, and that the object carries the value.

In addition to ordinary dependency among statements (Tip, 1995), we introduce new kinds of dependency among operations and statements. A get operation on a persistent variable depends on the operation that assigned the got value to the persistent variable. A method invocation whose receiver is not null executes the method body bound at runtime based on the receiver class. Thus method receivers work in a similar way to operands of conditional branching statements.

A value carrier itself may have been carried by another object, which is the carrier of the carrier of the value. We may further get the carrier of the carrier of the carrier of the value. For a carried value, we can thus obtain a sequence of references of persistent variables which have brought the value. We call such a sequence a reference path to the value or its carrier.

Application Frameworks

We categorize classes and methods in a framework application. We call classes (methods) contained in a framework framework classes (framework methods). We call those classes (methods) other than framework classes application-specific classes (application-specific
A Conceptual Descriptive-Comparative Study of Models and Standards of Processes in SE, SwE, and IT Disciplines Using the Theory of Systems
www.igi-global.com/chapter/conceptual-descriptive-comparative-study-models/38179?camid=4v1a

Industrial Applications of Emulation Techniques for the Early Evaluation of Secure Low-Power Embedded Systems
www.igi-global.com/chapter/industrial-applications-of-emulation-techniques-for-the-early-evaluation-of-secure-low-power-embedded-systems/116116?camid=4v1a