2007

Sofya: Supporting Rapid Development of Dynamic Program Analyses for Java

Alex Kinneer
University of Nebraska-Lincoln, akinneer@cse.unl.edu

Matthew B. Dwyer
University of Nebraska-Lincoln, dwyer@cse.unl.edu

Gregg Rothermel
University of Nebraska-Lincoln, grothermel2@unl.edu

Digital Object Identifier: 10.1109/ICSECOMPANION.2007.68
Publication Year: 2007 , Page(s): 51 - 52

Sofya: Supporting Rapid Development of Dynamic Program Analyses for Java *

Alex Kinneer, Matthew B. Dwyer, Gregg Rothermel
Department of Computer Science and Engineering
University of Nebraska - Lincoln
{akinneer,dwyer,grother}@cse.unl.edu

Abstract

Dynamic analysis is an increasingly important means of supporting software validation and maintenance. To date, developers of dynamic analyses have used low-level instrumentation and debug interfaces to realize their analyses. Many dynamic analyses, however, share multiple common high-level requirements, e.g., capture of program data state as well as events, and efficient and accurate event capture in the presence of threading. We present SOFYA *– an infra-structure designed to provide high-level, efficient, concurrency-aware support for building analyses that reason about rich observations of program data and events. It provides a layered, modular architecture, which has been successfully used to rapidly develop and evaluate a variety of demanding dynamic program analyses. In this paper, we describe the* SOFYA *framework, the challenges it addresses, and survey several such analyses.*

1. Introduction

A wide variety of techniques reported in the literature, e.g. [2, 6], use observations collected during actual runs of Java programs to perform analyses for verification and validation; new techniques are being reported frequently. Most of these techniques are sensitive to both the *accuracy* of the program observations – how faithfully the reporting of observed events reflects the actual ordering of those events in the monitored program – and the *efficiency* with which those observations can be delivered. Analyses that receive incorrectly ordered events may produce wrong results, and analyses that cannot process a large volume of events efficiently may simply run too slowly to be of any use. We discuss common problems encountered in implementing such analyses and describe how the SOFYA [5] dynamic analysis infrastructure addresses those problems.

Specification of program observations. Developers need to specify the program observations relevant to their analysis. It is generally accepted that modifying source code by hand for each analysis is too costly and error prone. Intermediary technologies, on the other hand, may present complex APIs that are difficult for the analyst to learn to use effectively, or that may not allow the analyst to describe the observation and the associated payload data, e.g., receiver object identity, needed for the analysis. SOFYA provides an expressive, but simple, declarative language for describing observations and associated payloads and automates the generation of code to capture them at run-time.

Efficient event capture. Using instrumentation and debugger connections to capture program observations introduces overhead. The literature reports multiple overhead reduction techniques, but, our experience indicates that developers do not generally apply these best-practices in their analysis implementations. SOFYA relieves analysis developers of the need to work with complex libraries and tools, such as the Bytecode Engineering Libary (BCEL), and enables the reuse of robust, efficient, and concurrency-safe strategies for the capture of a broad set of observations. It also provides several novel performance enhancements, such as support for dynamically modifying the set of active observations during analysis [1].

Accurate event capture. Concurrency can lead to hard to find bugs; consequently, many validation and verification techniques, e.g. [6], are aimed at detecting concurrency related errors. Without additional synchronization, which slows the program, byte code instrumentation cannot guarantee that the order of observed events corresponds with the order of occurrence of those events in the program. Synchronization introduced by instrumentation can interfere with the natural scheduling of threads in a monitored program, leading to questionable results from analyses investigating effects of thread ordering. While an efficient and correct solution to this problem is difficult to achieve, SOFYA includes a number of strategies that reduce the risk of imprecise reporting without losing efficiency.

Event processing. Many analyses need to distinguish between events occurring on different object instances. Since it is generally not possible to know the identity or even the

*This work was supported in part by the National Science Foundation through awards 0429149, 0444167, 0454203, and 0541263.

number of instances of an object statically, this presents a real challenge to analysis developers. SOFYA implements a publish-subscribe architecture that supports flexible filtering and routing of observations. Streams of correlated observations, such as those sharing the same receiver object, are routed to subscribing analysis components. These streams can be generated and re-routed dynamically as the program under analysis executes. SOFYA's standard architecture allows modularization of event processing in specific analyses while supporting flexible creation and combination of analysis components on-the-fly.

Many new dynamic analyses reported in literature are evaluated with a specialized implementation, even though they often share event capture requirements with existing techniques; such implementations are often described as prototypes. This leads to redundant and inefficient tools, which in our experience often suffer from common errors and shortcomings. It also inhibits the comparative evaluation of techniques. SOFYA addresses this by providing a common framework that frees developers from the burden of repeatedly dealing with these challenges so that they can rapidly implement and evaluate novel analysis techniques.

2. Sofya Architecture

Sofya's event capture components are organized into a layered architecture, which is presented in detail at [5]. The top layers present a programmatic publish/subscribe API targeted at "client" program analyses. This layered architecture factors out different aspects of event capture and dispatching so that they can evolve over time to track technology advances or be customized for a specific analysis without affecting existing analysis clients. We provide a brief summary of each layer.

Layer 1. Provides information to guide the activities of other layers, e.g, processing of observables defined in the Event Description Language (EDL). EDL is used by Sofya to specify events, and associated data values, to be captured from a class of "semantic" events, such as, method invocation, field read/write and lock acquire/release.

Layer 2. Capture of observations is achieved either through instrumentation, defined in this layer, or through debugger interface support (Layer 3). Sofya provides highly optimized and robust bytecode instrumentors in this layer, built using BCEL, to capture various program observations.

Layer 3. Provides communication mechanisms to transfer information from instrumentation and debugger interfaces to event dispatchers (Layer 4). Sofya runs monitored programs in their own virtual machine, which prevents event processing from interfering with program behavior.

Layer 4. Implements *event dispatchers - publishers* of captured program events. EDL can be used to ensure that only selected events are delivered to a given analysis client.

Layer 5. Provides splitters, filters, and routers to manipulate event streams published by event dispatchers. A splitter breaks a single event stream into multiple streams based on some criteria, such as thread ID. Filters are used to select particular events of interest out of an event stream.

Most dynamic program analyses can be rapidly implemented using the services provided by layers 4 and 5, thus benefiting from the carefully engineered efficiency and correctness of the lower layers without incurring the difficulty or cost of implementing that functionality.

3. Experience and Conclusions

We have used SOFYA to implement a wide-variety of different dynamic analyses including: multi-lockset race detection [4], vector-clock happens-before analysis [4], dynamic escape analysis [3], atomicity [6], variants of sequencing property inference analyses [7], and a number of sophisticated variants of temporal property conformance checkers [1]. Our experience across these 6 distinct classes of analyses, for which we have implemented a total of 13 different variants, can be contrasted to our experience building dynamic analyses using low-level libraries like BCEL directly over the past several years. Analyses built using SOFYA can be implemented more quickly, and are, at least, competitive in terms of overhead. We believe that SOFYA's standard interfaces and flexible publish-subscribe approach for connecting analysis components will improve analysis developer productivity by enabling the creation and reuse of high-level analysis building block components. SOFYA supports analysis developers by providing an efficient and well-engineered framework for rapidly implementing, evaluating, and comparing dynamic program analysis techniques. The SOFYA website [5] provides current information on the development of SOFYA, and allows free access to current versions of SOFYA.

References

[1] M. B. Dwyer, A. Kinneer, and S. Elbaum. Adaptive online program analysis. In *Int'l. Conf. Softw. Eng.*, 2007 (to appear).

[2] K. Havelund and G. Roşu. An overview of the runtime verification tool Java PathExplorer. *Formal Meth. Sys. Design*, 24(2):189–215, 2004.

[3] H. Nishiyama. Detecting data races using dynamic escape analysis based on read barrier. In *Proc. Virt. Machine Research and Tech. Symp.*, pages 127–138, 2004.

[4] R. O'Callahan and J.-D. Choi. Hybrid dynamic data race detection. In *Symp. Princ. Prac. Par. Prog.*, 2003.

[5] http://sofya.unl.edu.

[6] L. Wang and S. D. Stoller. Runtime analysis of atomicity for multi-threaded programs. *IEEE Trans. Softw. Eng.*, 32:93–110, Feb 2006.

[7] W. Weimer and G. Necula. Mining temporal specifications for error detection. In *Conf. Tools Alg. Constr. Anal. Sys.*, pages 461–476, April 2005.

0-7695-2892-9/07 $20.00