2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE) | 978-1-6654-0337-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/ASE51524.2021.9678588

Business Process Extraction Using Static Analysis

Md Rofiqul Islam
Dept. of Computer Science
Baylor University
Waco, Texas, United States
rofiqul_islam1@baylor.edu

Tomas Cerny
Dept. of Computer Science
Baylor University
Waco, Texas, United States
tomas_cerny@baylor.edu

***Abstract*—Business process mining of a large-scale project has many benefits such as finding vulnerabilities, improving processes, collecting data for data science, generating more clear and simple representation, etc. The general way of process mining is to turn event data such as application logs into insights and actions. Observing logs broad enough to depict the whole business logic scenario of a large project can become very costly due to difficult environment setup, unavailability of users, presence of not reachable or hardly reachable log statements, etc. Using static source code analysis to extract logs and arranging them perfect runtime execution order is a potential way to solve the problem and reduce the business process mining operation cost.**

***Index Terms*—Static Code Analysis, Log analysis, Business Process Mining, Distributed Systems, Cloud Computing**

I. Introduction

Static source code analysis is a method of automatically examining the source code against a set of coding rules which generally addresses weaknesses and vulnerabilities [1]. On the other hand, logs are automatically created to record all events from a project that can reflect the whole business process [2]. However, to get the log file, we need to run the project. Sometimes running a project can be very costly, which can require a difficult platform and database setup. Moreover, we need to generate a large set of input to test the code for all business logic rules. This problem can be solved by constructing a special static Source Code Analyzer (SCA), which can find out all log messages from a project code and sort them in perfect order as they would print at runtime. This brings the opportunity to understand the business process directly from static code analysis. Such a process is more economically beneficial than extracting business rules from the application log. To obtain the log messages in the expected order, we have implemented, extended, and improved the existing proof of concept tool [3] for business process extraction. To demonstrate our approach and assess its precision, we demonstrate a case study using a distributed system. Our results indicate the suitability of the approach for quick extraction of business processes, driven by static code analysis.

II. Background and Related work

Static code analysis can examine program source code and try to generate all possible behaviors that might occur at runtime execution [4]. There are many tools available for static source code analysis for different programming languages [5]. Most of those tools are mainly targeted to a generalized representation of source code. For static log analysis, it is necessary to modify the static source code analyzer. PROF [3] is such a tool that is actually a non-intrusive request flow profiler and has a special feature of log analysis. It's mainly focused on distributed infrastructure.

Business Process Mining (BPM) is a technique to extract business logic from business events such as event logs [6]. BPM can discover much information such as process, data, control, logic, organizational and social structures from these event logs [7]. Different tools regarding BPM are available on the market, and one of the most popular tools is PROM tool [7], [8]. Prom tool provides a multidimensional representation of extracted business logic after the mining process.

Generally, BPM reconstruction approaches work with application logs, and thus, combining static log analysis with BPM provides an alternative approach. Static code log analysis has some limitations, and this work finds out how it affects the BPM reconstruction.

III. Design strategy of BPM SCA

SCA builds upon a library for parsing source code [9], in our case Java. It searches through identified constructs, typically class, to finds out variables, dependencies, method bodies, if-else conditions, loops, annotations, etc. The steps of a design are given below:

Creation of class tree: A class tree is a graph representation of the relationship between multiple classes under the same project. First, all classes are identified under the project directories and then parsed. In Java, each class's import declarations and annotation declarations enable the creation of a relation graph tree between all classes.

Creation of method model tree: Method models are representations of methods in a project which contains *method id, method name, class name, in method variable list, invoked method list, parameter list, log list, etc.*. Method model tree is a graph representation of the relation between different methods and can be traversed to generate the execution order of the methods. Method model tree can be generated based on the invoked method list of each method model class object because this attribute is actually the manager of relations between multiple methods.

This material is based upon work supported by the National Science Foundation under Grant No. 1854049 and a grant from Red Hat Research, https://research.redhat.com.

978-1-6654-0337-5/21/$31.00 ©2021 IEEE
DOI 10.1109/ASE51524.2021.00135

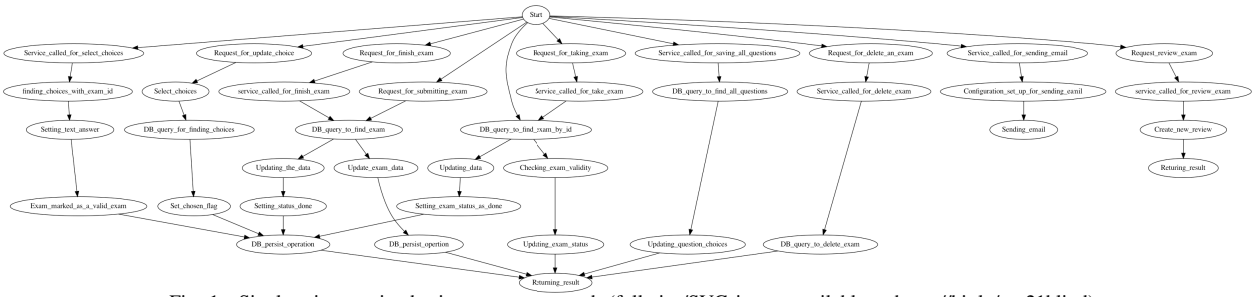

Fig. 1. Single microservice business process graph (full size/SVG image available at https://bit.ly/ase21blind)

Merging and Sorting Invoked Methods and Log Printing Statements: To arrange the logs in the order of execution at runtime, only two possible events need to be considered. The first incident is when a log printing statement is reached, and the second incident is when a method invocation statement is reached. When the first event happens, the log print statement is added to the output, but when the second event is reached, the called method needs to be traversed first and then go back to the previous method to complete the process. The sequence of these events depends on the line number where they occur.

Finding top-level method: Before starting the traverse on the method model tree, the starting point (root nodes) needs to be identified. The methods which are not called from any other method are those nodes and are indicated as top-level methods. They can be identified from the method tree by the topological sort [10] through a DFS (Depth First Search) [11] on the method tree with sorting based on the traverse complete time of each node.

Graph Traverse and Arrange the logs in runtime order: Before starting the traversing process, two important things needs to be discussed – *cases* and *events*. A case is a traversing of a branch in method tree from a specific top method and returning to it again. Events are existing nodes in a branch of a particular traverse. Traverse will be started from every top-level method one by one with a newly generated case id. An iteration starts through the combined list of logs and invoked method of the top-level methods; if any log printing statement is found, then a new event id is generated and written into the output file with the case id and event id. If any method invoking statements is found, then the processing of the current method is paused and starts work with the new invoked method by iterating through the combined list of this new invoked method same as above.

Generating Business logic graph: Based on the cases and events in the output file, there are two ways to generate the business logic graph. In first way, and existing tool like PROM [8] can be used and in second way, custom program can be developed for this purpose.

IV. Case Study Evaluation

For performance evaluation, we have used a micro-service project Teacher Management System (TMS) as testbed. It is built with Java programming language using SpringBoot framework [12] with log4j logging mechanism [13]. This testbed follows the repository pattern and service layer pattern, which consists of controllers, services, repositories, and models. The service layer is the holder of all business logic in this architecture.

First, we have generated a business process graph from our testbed with our proposed SCA tool. Then we create a distributed system environment with Docker [14], and Kubernetes [15] and deployed the testbed's microservices on different docker instances and run the system generating sufficient application logs for our evaluation.

Number of Microservices	6
Number of cases found in application log	153
Number of cases generated from BPM graph	167
Number of cases match	131
Precision of BPM graph	79%
Recall of BPM graph	86%

We collected application logs from different docker instances and sorted them according to the event triggering time. After that, we have compared our log file with the business process mining graph from each micro-service. In this comparing process, we have collected information on many subjects, such as which business processes are requested the most, which business logic has vulnerabilities, our BPM accuracy using static log analysis, cost of each user request, etc. Figure 1. shows the BPM graph of a single microservice. Our testbed consists of 6 microservices, and we obtained a different BPM graph for each, then we connected all 6 graphs to generate the full BPM graph for the whole system and collected the necessary data for evaluation.

V. Conclusion and Future Work

This paper demonstrates how static analysis can be used to extract the log messages from a system code and cluster the messages to derive business processes. It provides an alternative to dynamic analysis. However, since some portion of code manifests itself only at runtime, such as polymorphic method override, inheritance, etc., there is a small impact on the accuracy of the static analysis approach. Still, it provides a great opportunity to understand the business logic embedded in a system without running it. This provides a great trade-off between a small impact on accuracy and running a large-scale system to generate logs to depict the whole business process set, which is costly. In future work, we target a better prediction of the runtime system behavior to increase the accuracy of business process mining.

Our tool is available at: https://github.com/Rofiqul-Islam/logparser

REFERENCES

[1] P. Louridas, "Static code analysis," *IEEE Software*, vol. 23, no. 4, pp. 58–61, 2006.

[2] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *2009 Ninth IEEE International Conference on Data Mining*, 2009, pp. 149–158.

[3] X. Zhao, Y. Zhang, D. Lion, M. F. Ullah, Y. Luo, D. Yuan, and M. Stumm, "lprof: A non-intrusive request flow profiler for distributed systems," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, Oct. 2014, pp. 629–644. [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/zhao

[4] I. Gomes, P. Morgado, T. Gomes, and R. Moreira, "An overview on the static code analysis approach in software development," *Faculdade de Engenharia da Universidade do Porto, Portugal*, 2009.

[5] J. Novak, A. Krajnc *et al.*, "Taxonomy of static code analysis tools," in *The 33rd international convention MIPRO*. IEEE, 2010, pp. 418–422.

[6] W. Van Der Aalst, "Process mining," *Commun. ACM*, vol. 55, no. 8, p. 76–83, Aug. 2012. [Online]. Available: https://doi.org/10.1145/2240236.2240257

[7] W. van der Aalst, H. Reijers, A. Weijters, B. van Dongen, A. Alves de Medeiros, M. Song, and H. Verbeek, "Business process mining: An industrial application," *Information Systems*, vol. 32, no. 5, pp. 713–732, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437906000305

[8] B. F. Van Dongen, A. K. A. de Medeiros, H. Verbeek, A. Weijters, and W. M. van Der Aalst, "The prom framework: A new era in process mining tool support," in *International conference on application and theory of petri nets*. Springer, 2005, pp. 444–454.

[9] "VoidVisitorAdapter - javaparser-core 3.3.1 javadoc." [Online]. Available: https://javadoc.io/doc/com.github.javaparser/javaparser-core/3.3.1/com/github/javaparser/ast/visitor/VoidVisitorAdapter.html

[10] C. Pang, J. Wang, Y. Cheng, H. Zhang, and T. Li, "Topological sorts on dags," *Information Processing Letters*, vol. 115, no. 2, pp. 298–301, 2015.

[11] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[12] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch, A. Wilkinson, M. Overdijk, C. Dupuis, and S. Deleuze, "Spring boot reference guide," *Part IV. Spring Boot features*, vol. 24, 2013.

[13] S. Gupta, "log4j and j2ee," *Pro Apache Log4j*, pp. 157–161, 2005.

[14] C. Anderson, "Docker [software engineering]," *Ieee Software*, vol. 32, no. 3, pp. 102–c3, 2015.

[15] D. Vohra, *Kubernetes microservices with Docker*. Apress, 2016.