The Concept of Trace Summarization*

Abdelwahab Hamou-Lhadj
University of Ottawa
800 King Edward Avenue
Ottawa, Ontario, K1N 6N5 Canada
ahamou@site.uottawa.ca

Abstract

Recently, trace analysis techniques have gained a lot of attention due to the important role they play in understanding the system behavioral aspects. However, manipulating execution traces is still a tedious task despite the numerous techniques implemented in existing trace analysis tools. The problem is that traces are extraordinary large and abstracting out their main content calls for more advanced solutions. In this paper, I introduce the concept of trace summarization as the process of taking a trace as input and returning a summary of the main invoked events as output. A discussion on how text summarization techniques can be applied to summarizing the content of traces is presented.

Keywords:

Analysis of program execution, Program analysis for program understanding, Dynamic Analysis, Reverse Engineering.

1. Introduction

Dynamic analysis is crucial for understanding the behavior of a software system. Understanding an object-oriented (OO) system, for example, is not easy if one relies only on static analysis of the source code [15]. Polymorphism and dynamic binding, in particular, tend to obscure the relationships among the system artifacts.

Run-time information is typically represented using execution traces. Although, there are different kinds of traces, this paper focuses on traces of routine calls. I use the term routine to refer to a function, a procedure, or a method in a class.

Many studies such as the ones presented by Systä [14], Zayour [17], Lange et al. [8], and Jerding et al. [6] have shown that, if done effectively, trace analysis can help with various reengineering tasks such redocumenting the system behavior, maintaining the system, or simply understanding the implementation of software features.

However, the large size of traces poses serious limitations to applying dynamic analysis. To address this issue, most existing solutions provide a set of fine-grained operations embedded into tools that software engineers can use to go from a raw sequence of events to a more understandable trace content [6, 8, 14, 17]. But due to the size and complexity of typical and most interesting traces, this bottom-up approach can be difficult to perform.

In addition, software engineers who have some knowledge of the system and the domain will most likely want to have the possibility to perform a top-down analysis of the trace – They want to have the ability to look at the 'big picture' first and then dig into the details. Many research studies in program comprehension have shown that an adequate understanding of the system artifacts require usually both approaches (i.e. bottom-up and top-down) [12].

In this paper, I discuss the concept of trace summarization, which is a process of taking an execution trace as input and return a summary of its main content as output. This is similar to text summarization where abstracts can be extracted from large documents. Using an abstract, the reader can learn about the main facts of the document without having to read entirely its content.

Trace summaries can be used in various ways:

- Enable top-down analysis of execution traces, something that is not supported by most existing trace analysis tools.
- Recover the documentation of the dynamics of a software system that suffers from poor to non-existent documentation.
- Uncover inconsistencies that may exist between the way the system is designed and its implementation. This can be achieved by

* This research is supported by Natural Sciences and Engineering Research Council of Canada (NSERC)

comparing the extracted models to the models created during the design phase [6, 11]. The analysis of these inconsistencies can help determine areas of the system that need reengineering.

The rest of this paper is organized as follows: In the next section, I discuss trace summarization from the perspective of text summarization techniques and show the similarity between the two fields. In Section 3, I discuss how a summary can be validated.

Most of the concepts presented in this paper are still fresh ideas that constitute an ongoing research. They will need to be validated in the future.

2. What is Trace Summarization?

In general, a text summary refers to an abstract representing the main points of a document while removing the details.

Jones [7] defines a summary of a text as "a derivative of a source text condensed by selection and/or generalization on important content". Similarly, I define a summary of a trace as an abstract representation of the trace that results from selecting the main content by both selection and generalization.

Although, this definition is too specific to be used to define a summary of a trace, it points towards several interesting questions that deserve further investigation. These are: what would be a suitable size for the summary? And how should the selection and generalization of important content be done?

2.1 Adequate Size of a Summary

While it is obvious that the size of a summary should be considerably smaller than the size of the source document, it seems unreasonable to fix the summary's size in advance.

In fact, a suitable size of a summary of a trace will depend in part upon the knowledge the software engineer has of the functionality under study, the nature of the function being traced and the type of problem the trace is being used to solve (debugging, understanding, etc.). This suggests that any tool should allow the summary to be dynamically expanded or contracted until it is right for the purpose at hand. I suggest that no matter how large the original trace, there will be situations when a summary of less than a page will be ideal, and there will be situations where a summary of several thousand of lines may be better.

2.2 Content Selection

In text summarization, the selection of important content from a document is usually performed by ranking the document phrases according to their importance. Importance is measured using various techniques. In what follows, I present the most classical techniques and discuss their applicability to trace summarization.

Perhaps, the most popular technique for building text summaries is the word distribution method [4, 9]. This method is based on the assumption that the most frequent words of a document represent also its most important concepts. Once the word frequencies are computed, the document phrases are ranked according to the number of the most frequent words they contain. Similarly, one possible way of selecting the most important events from a trace is to examine their frequency distribution.

In fact, frequency analysis has also been used in various contexts of dynamic analysis. Profiles, for example, use the number of times specific events are executed to enable software maintainers prevent performance bottlenecks. In [1], Ball introduces the concept of Frequency Spectrum Analysis which is a technique that aims to cluster the trace components according to whether they have similar frequencies or not. This can help recover the system architecture.

However, the application of frequency analysis to select important events from execution traces raises several issues. First, the fact that traces contain several repetitions due to the presence of loops and recursion in the source code might render the results of frequency analysis inaccurate. For example, there is no evidence that something that is called ten times due to a loop would be more or less important than a routine that is called once or twice just because it did not happen to be in a repetitive code. Second, something that is repeated several times in one trace might not have the same behavior in another trace. Finally, our experience with using traces has shown that even if we remove the most frequent event from traces, traces will still be very large for humans to understand, which might make this technique useful but far from sufficient.

Another text summarization technique is the cue phrases method, which is based on the idea that most texts contain phrases that can lead to the identification of important content (e.g., "in conclusion", "the paper describes", etc) [4]. Similarly, the routine names can be used to extract important routines assuming that the system follows strict naming conventions. For example, during the exploration of a trace generated from a system that implements the C4.5 classification algorithm [16], my colleagues and I found that many routines are actually named according to the various steps of the algorithm

such as buildClassifier, buildTree, etc. The 'cue routines (or events)' technique is certainly a powerful approach for building summaries from traces. However, in order to be successful, it requires having a system that follows some sort of naming conventions. In addition to this, there is a need to deal with the various naming matching issues that might occur. For example, some routine names might use acronyms or short names which might complicate the matching process.

The third text summarization technique discussed in this paper is the location of phrases in the document [2]. The idea is that the position of sentences in a document can be an indicator of how important they are. In text summarization, the first and last phrases of a paragraph are usually the ones that convey the most relevant content.

When applied to traces, we need to investigate whether the location of routines in the call tree (i.e. trace) can play a relevant role in determining their importance. There are certainly situations where this can be valid. For example, if the system is designed according to a layered architecture then the bottom layers are perhaps the ones that are the least important since they implement the system low-level details. These usually appear in the call tree as leaf nodes.

Some thoughts: a trace can be viewed according to two dimensions: vertical and horizontal dimensions as shown in Figure 1. The vertical dimension reflects the sequential nature of the execution of the system. One possible scenario for applying the location technique is based on the ability to partition the trace into smaller sequences that depict different behavioral aspects of the system, and then select the first calls of each sequence and add them to the summary. This is like having a text composed of many sequential paragraphs and that the summarizer needs to visit each of them. It is obvious that in practice this might not be easy to perform. Indeed, the partitioning of a trace might be challenging. And even if it is done successfully, we might end up having a considerably large number of partitions where some of them do not necessarily convey the most important content.

The horizontal dimension focuses on the fact that a trace is viewed as a tree structure containing many levels of calls. The idea is to develop a level analysis technique in order to detect the levels that introduce trace components used as mere implementation details. For example, the routines that appear always in the first levels of the tree might represent the system high-level concepts whereas the ones that appear at all levels might be utilities (because they are called by many other routines).

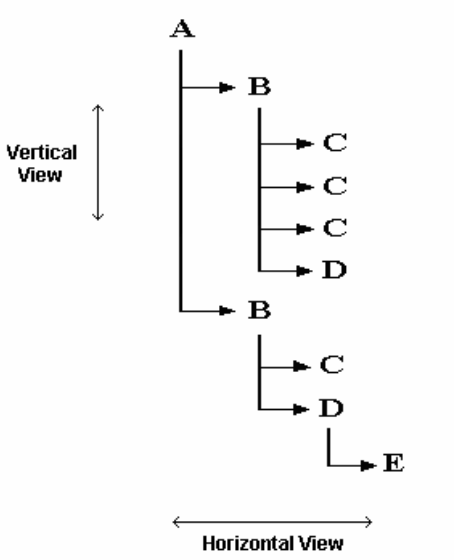

Figure 1. The vertical and horizontal views of a call tree

2.3 Content Generalization

Content generalization consists of generalization of specific content with more general abstract information [7]. When applied to execution traces, generalization can be performed in two ways:

The first approach to generalization involves assigning a high-level description to selected sequences of events. For example, many trace analysis tools provide the users with the ability to select a sequence of calls and replace it with a description expressed in a natural language. However, this approach relies on user input and would be very hard to automate.

A second approach to generalization relies on treating similar sequences of execution patterns as if they were the same. This approach can be automated by varying the similarity function. For example, in the simplest case all sequences with the same elements, ignoring order, could be treated the same. Or, all subtrees that differ by only a certain edit distance could be treated the same. All trace summarization approaches will need to use this technique to some extent.

For example, the call tree of Figure 1 can be summarized into the tree shown in Figure 2 by ignoring the number of contiguous repetitions of the node labeled 'C' and by comparing subtrees up to level 2 (this will ignoring the node 'E'). A discussion on how matching criteria can be used to reduce the size of a trace is presented by De Pauw et al. [3].

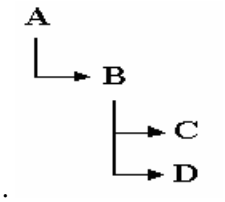

Figure 2. A summary extracted from the tree of Figure 1 by applying generalization

However, it might be hard to determine how the matching criteria should be combined in order to extract the most meaningful content. Different combinations will most likely result in different summaries. Tools that support the generation of summaries will need to allow enough flexibility to apply the matching criteria in several ways.

3. Validating a Trace Summary

Perhaps, one of the most difficult questions when evaluating a summary is to agree about what constitutes a good summary. In other words, what distinguishes good summaries from bad summaries (assuming that there are bad summaries)?

In text summarization, there are two techniques for evaluating summaries: extrinsic and intrinsic evaluation. The extrinsic evaluation is based on evaluating the quality of the summary based on how it affects the completion of some other tasks [5]. The intrinsic evaluation consists of assessing the quality of the summary by analyzing its content [10]. Using this approach, a summary is judged according to whether it conveys the main ideas of the text or not, how close it is to an ideal summary that would have been written by the author of the document, etc.

Extrinsic evaluation of a trace summary will typically involve using summaries to help with various software maintenance tasks such as adding new features, fixing defects, etc.

The intrinsic evaluation technique can be used to assess whether the extracted summary reflect a high-level representation of the traced scenario that would be similar to the one that a software engineer would design. In practice, I suspect that both types of evaluations are needed.

4. Conclusions and Future Directions

The objective of this paper is to present a technique for analyzing traces based on summarizing their main content. This technique is referred to as Trace Summarization, which the process of taking a trace as input and generating an abstract of its main content as output. I argued that summaries can be very useful to software engineers who want to perform top-down analysis of a trace, understand the system behavior, or uncover inconsistencies between the system design and its actual implementation.

In the paper, a discussion on how text summarization techniques can be applied to extracting summaries from trace is presented.

Future directions should focus on examining the techniques presented in this paper in more detail including experimenting with several traces. The experiments should take into account systems of different domains, the expertise software engineers have of the system, and the type of software maintenance performed.

References

[1] T. Ball, "The Concept of Dynamic Analysis", *ACM Conference on Foundations of Software Engineering (FSE)*, September 1999

[2] P. Baxendale, "Machine-made index for technical literature – an experiment", *IBM. Journal of Research and Development* 2:354-361, 1958

[3] W. De Pauw, D. Lorenz, J. Vlissides, M. Wegman, "Execution Patterns in Object-Oriented Visualization", In *Proc. of the 4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS)*, Santa Fe, NM, 1998, pp. 219-234

[4] H. Edmundson, "New methods in automatic extracting", *Journal of the ACM* 16(2): 264-285, 1969

[5] H. R. Jing, K. McKeown, and M. Elhadad, "Summarization evaluation methods: Experiments and analysis", *In Working Notes of the AAAI Spring Symposium on Intelligent Text Summarization*, 1998, pp. 60-68

[6] D. Jerding, S. Rugaber. "Using Visualization for Architecture Localization and Extraction", In *Proc. of the 4th Working Conference on Reverse Engineering*, Amsterdam, Netherlands, October 1997

[7] S. K. Jones, "Automatic summarising: factors and directions", In *Advances in Automatic Text Summarization, MIT Press*, 1998, pp. 1-14

[8] D. B. Lange, Y. Nakamura, "Object-Oriented Program Tracing and Visualization", *IEEE Computer*, 30(5), 1997, pp. 63-70

[9] H. Lunh, “The Automatic Creation of Literature Abstracts”, *IBM Journal of Research and Development 2(2):* 159-165, 1958

[10] C. Paice, and P. Jones, “The identification of Important Concepts in Highly Structured Technical Papers”, In Proc. of the 16th Annual International ACM SIGR Conference on research and Development in Information retrieval, 1993, pp. 69-78

[11] Reiss S. P., Renieris M., “Encoding program executions”, In *Proc. of the 23rd international conference on Software Engineering*, Toronto, Canada, 2001, pp. 221-230

[12] M.A. Storey, K. Wong, H.A. Müller, “How do Program Understanding Tools Affect how Programmers Understand Programs?”, In *Proc. of the 4th Working Conference on Reverse Engineering*, 1997, pp. 183 - 207

[13] T. Strzalkowski, G. Stein, J. Wang, B. Wise, “Robust Practical Text Summarization”, In *Advances in Automatic Text Summarization*, MIT Press, 1999

[14] T. Systä, “Understanding the Behaviour of Java Programs”, In *Proc. of the 7th Working Conference on Reverse Engineering (WCRE)*, Brisbane, QL, 2000, pp. 214-223

[15] N. Wilde, R. Huitt, “Maintenance Support for Object-Oriented Programs”, *Transactions on Software Engineering*, 18(12):1038–1044, Dec. 1992

[16] Witten I. H., Frank E. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, 1999

[17] I. Zayour, “Reverse Engineering: A Cognitive Approach, a Case Study and a Tool”, Ph.D. dissertation, University of Ottawa, 2002