A Software Behaviour Analysis Framework Based on the Human Perception Systems (NIER Track)

Heidar Pirzadeh and Abdelwahab Hamou-Lhadj
Software Behaviour Analysis Lab
Department of Electrical and Computer Engineering
Concordia University
{s_pirzad, abdelw}@ece.concordia.ca

ABSTRACT

Understanding software behaviour can help in a variety of software engineering tasks if one can develop effective techniques for analyzing the information generated from a system's run. These techniques often rely on tracing. Traces, however, can be considerably large and complex to process.
In this paper, we present an innovative approach for trace analysis inspired by the way the human brain and perception systems operate. The idea is to mimic the psychological processes that have been developed over the years to explain how our perception system deals with huge volume of visual data. We show how similar mechanisms can be applied to the abstraction and simplification of large traces. Some preliminary results are also presented.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement- *Restructuring, reverse engineering, and reengineering*

General Terms

Algorithms, Experimentation, Human Factors, Theory

Keywords

Program Comprehension, Dynamic Analysis, Trace Analysis, Human Perception, Psychological Processes, Phase Detection.

1. INTRODUCTION

Since the outset of our research we have been looking for ways to help software engineers understand the behavioural aspects of software systems through tracing and run-time monitoring techniques. This can be useful in a number of software engineering activities including maintenance, performance analysis, and most recently security.

Understanding software behavior, however, is a difficult task. Tracing even a small system can generate large amounts of data that pose a real obstacle to any viable analysis. We (and other researchers) have been studying, for a long time now, techniques to reduce the size of large traces while keeping their content. The common approach has been based on investigating various heuristics that can guide the trace abstraction and simplification process. Although significant improvement has been made in the area, there is a consensus within the trace analysis community that more needs to be done.

In this paper, we present what we think it is a novel approach to the problem by drawing parallels between the way software systems behaviour is analyzed and the way the human brain operates when dealing with information received through the visual sense. This is motivated by the following three observations:

- The amount of visual data received through our sensory is in general too high to be completely processed in detail [3] so is the amount of information generated from a system run.
- The inability for the human brain to keep track of all relevant data in a specific domain of interest in both cases. There is a limited amount of information that can be handled by the human memory at any given time. George Miller showed that short-term memory, or working memory, has a limited capacity (only 7±2 pieces of information, such as words or numbers, can be held at any one time) and cannot keep track of all the information from the visited knowledge domain [2].
- The need to acquire the needed information in a relatively short time. In the case of trace analysis, time-to-market and other constraints such as the criticality of the analysis makes it necessary to obtain the needed information as quickest as possible. The same holds for human perception; our perception system works in a way we grasp the essence of a scene within a small fraction of a second [3]. This remarkable speed in gaining the information contributes to low response time and quick reactions.

There are a number of processes that are proposed in psychology to explain how the human brain and the perception system automatically (not voluntarily) deal with huge volume of visual data considering limited short-term memory and necessity of a short response time. In this research, we aim to learn from these techniques and to build similar mechanisms that can help process large traces. The processes in questions are as follows [1, 3, 5, 8, 9, 13]:

1. It appears that our perceptual system segments local elements against their context and integrates them as objects and regions (implicit perception).
2. The segmented scene is then quickly scanned with eye movements so as the brain obtains an overall impression of it (global perception).
3. The scene is analyzed in more detail by visiting the regions in a certain order. The pop-out effect is an important factor in this process (preattentive process).

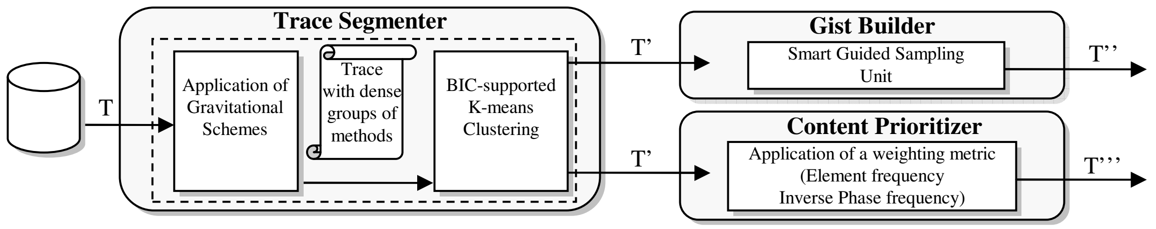
Trace Segmenter
T
Application of Gravitational Schemes
Trace with dense groups of methods
BIC-supported K-means Clustering
T'
Gist Builder
Smart Guided Sampling Unit
T''
T'
Content Prioritizer
Application of a weighting metric (Element frequency Inverse Phase frequency)
T'''

Figure 1. Our proposed framework and its components mimic the processes of human perception system

In this paper, we introduce a framework for trace analysis that is inspired by these three processes. This is still an on-going work. Some preliminary studies have been conducted and reported in this paper. The long-term goal is to build better trace analysis tools that, again, can help understand various aspects of a running system.

2. FRAMEWORK

Our proposed framework is currently composed of three components (see Figure 1). Each of these components mimics one of the processes employed by the human perception system to process visual data. We discuss each of these components in the following subsections.

2.1 The Trace Segmenter

The first component of our framework, the trace segmenter, aims to divide a large trace into meaningful segments that represent what we call execution phases. Examples of execution phases could be initializing variables, applying a specific algorithm, etc. These segments can significantly simplify the exploration of large traces by enabling software engineers to browse traces as a flow of execution phases instead of mere low-level events.

To achieve this ambitious goal, we turn to Gestalt laws of perception, which describe how people group similar items visually based on their perception [13]. Gestalt psychology is an application of physics to essential parts of brain physiology that help determine the type of processes that occur in the brain when we see a scene, and how our perceptual systems follow certain grouping principles (e.g., good continuation, proximity, and similarity properties of the elements) [13] to integrate the scene elements (i.e. objects and regions) as a whole and not just as points and lines. These laws explain how our perceptual system segments local elements against their context and integrates them as objects.

In our previous work [16], we have developed two gravitational schemes (more precisely the similarity and continuity schemes) based on Gestalt laws that are used as gravitational forces that yield the formation of dense groups of trace elements, which indicate the candidate execution phases. When dense groups are formed, we automatically identify the beginning and end of each phase using K-means clustering with BIC (Bayesian Information Criterion) support. When applied to a trace T (see Figure 1), the output consists of a trace T' where the execution phases are identified.

The effect of applying each of the two gravitational schemes we have developed (i.e. similarity and continuity schemes) is as follows:

Similarity scheme: By applying this scheme the elements in the trace are repositioned in a way that the distance between the similar elements is reduced (Figure 2 (a)).

Continuity scheme: The application of this scheme results in the repositioning of the elements in a way that the elements that are continuously invoked (and not returned as much) are made closer one to another to emphasize a trend in the execution of the program (Figure 2 (b)).

The two gravitational schemes that we have developed are also aligned with the fact that a phase change in an execution trace corresponds to a significant change in the pattern of some attributes of the elements in the trace over time. Therefore, our strategy can be seen as reducing the distances between the elements for which the characteristics can form a pattern specifying a phase. Again, this is similar to the way a human brain automatically processes points and lines to form objects and regions.

2.2 Gist of Traces

When we first look at a scene, we have the impression to see the entire scene, and rarely focus on the details. This representative image of the scene is provided through quick eye movements over different parts of the scene. This scanning provides us with the gist of the scene by performing a single short sampling that lasts in the order of 0.1 sec. Boothe explains that the sampling mechanism cannot be a dumb process, as “it would seriously limit our ability to maintain a high-fidelity perceptual database” [5]: A ‘smart’ guided sampling must be performed. This smart guided sampling is suggested to be evolutionarily advantageous because it can speed up information processing. Uchida et al. [1] suggest that processing limited low-level information from short chunks could facilitate rapid construction of global percept of scenes. A smart process could be one that performs sampling on each segment of the scene rather than on mere unstructured details of the scene.

Similarly, in the domain of trace analysis, regular and random sampling techniques have often been used to reduce the size of execution traces (e.g., [10, 14]). In general, sampling techniques are concerned with selecting a sample of a trace for analysis instead of analyzing the entire trace. However, finding the right sampling parameters can be a difficult task and even if some parameters work well for one trace, they might not work for another trace (even if generated from the same system) [14]. Also, since trace sampling is often not based on information about the trace (e.g., distribution of the trace elements, its homogeneous subsequence, outliers, etc.) it may result in a sample that is not representative of the original trace. In general, when the population on which we perform sampling is not homogeneous (i.e., it is made up of elements that are different than each other in sub-populations, and that each sub-population represents a group of similar elements), then random sampling might result in an unrepresentative sample [15]. It is a common situation for execution traces not to be homogeneous. The reason is that a trace is composed of a sequence of events where each subsequence

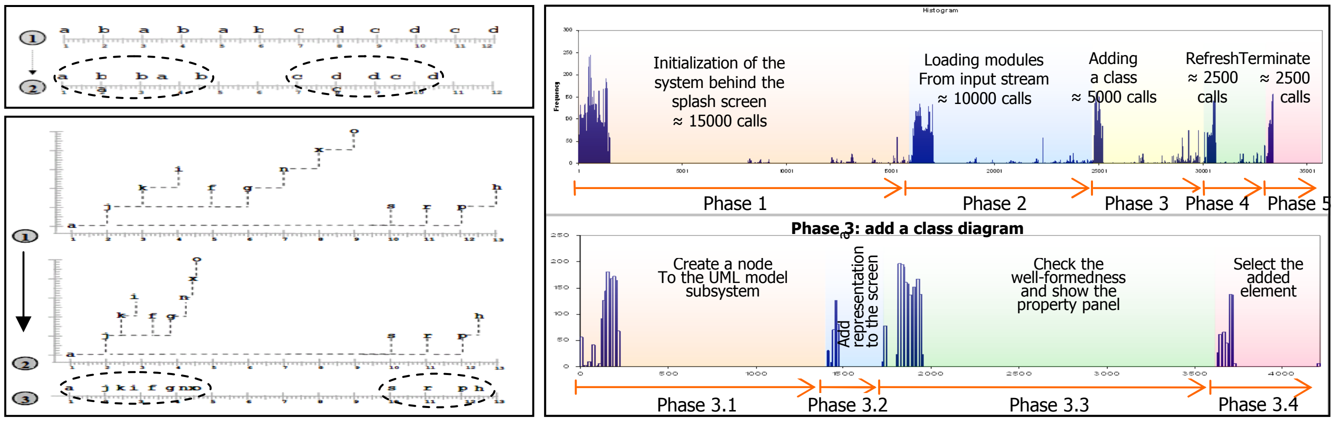

Figure 2. Left Top: The effect of similarity scheme on a sample trace, Left Bottom: effect of continuity scheme on a sample method call trace, Right Top: Major phases in adding a class diagram in ArgoUML, Right Bottom: sub-phases of Phase 3.

represents a specific task performed by the program. The events in one particular set of events can be completely different than the ones of another subsequence.

Inspired by the process of obtaining global percept in the human perception system, we propose a smart guided sampling process that makes use of proportional stratified sampling extensively studied in Information Theory [7]. In stratified sampling, first, the original trace needs to be separated into a desired number of non-overlapping and exhaustive subsets (called stratum) and then trace elements that would be part of the sample are selected within each stratum. The size of a sample of each stratum is in proportion with the population size of the stratum. By doing this, we guarantee that the final sample contains elements that are representative of every part of the trace.

As shown in Figure 1, we use the trace segments (execution phases) to serve as strata in the sampling process. In other words, once the phases have been detected (i.e. the trace T' is obtained – Figure 1), we start the stratified sampling process, which is implemented in our framework, as part of the smart guided sampling unit. This unit receives a phased execution trace T' as its input and outputs a sample of the execution trace using stratified sampling. Since the elements within each stratum are homogeneous, we perform the selection of trace elements from each stratum using random sampling. The size of a sample of each phase is in relative to the size of the phase. The result of this phase in a sampled trace T'' which is smaller trace than T' and yet representative of the content of T'.

It might sound contradictory that we are using random sampling, which was criticized earlier, to detect sample in each stratum. However, it should be noted that random sampling usually performs well on homogenous data (in this case the content of a stratum). As mentioned earlier, it does not return representative samples when applied to large non-homogenous data spaces such as an entire trace.

2.3 Prioritizing Content

The third process by which a human brain processes visual data is through analyzing the scene in more detail by visiting the regions in a certain order. The order could be selected by a preattentive process [8]. Pop-out effect is one of the important factors in this process that leads to rapid detection of elements that differ greatly from surrounding elements usually in single dimension such as color or orientation [9, 3]. In general, a high frequency of an element in one region shows the importance of that element while if one element is scattered between different regions is considered less important – an apple among oranges pops out. The elements that pop out exhibit relative importance as they have a unique perceptual feature. The basic assumption is that important elements should appear more times in one region and not in many other regions of a scene.

The question is therefore: what pops out in a trace? The idea of weighting trace elements based on their frequency has been proposed in many studies (e.g., [10, 11]) to detect trace elements that do not follow the same frequency distribution (they are invoked considerably more than the other elements). But simply relying on mere frequency, we do not think that it is sufficient to detect important elements of a trace. In fact, Durgerdil et al. [10] showed that elements that appear frequently all over the trace, called temporal omnipresent elements, are the least important elements.

We suggest a new technique similar to the preattentive prioritizing process in our perception system that takes into account both frequency of elements and their appearance in behavioral segments (execution phases) of the trace. In our technique, a weight is assigned to each trace element that represents the relative importance of that element in the whole trace. As shown in Figure 1, this is the role of the content prioritizer component, which receives a phased trace T' as input. The basic idea is that trace elements that appear often in a particular phase, but appear relatively infrequently in all the phases should receive the highest weight, meaning that it is doing something important in this particular phase. The unit generates a trace T''' containing only the important elements. The output of the content prioritizer is a set of the most relevant components that implement the traced feature. The important elements can also be used to create summaries from large traces. Because of the fact that our approach is similar, in principle, to the TFIDF (Term Frequency - Inverse Document Frequency) weighting approach in information retrieval [6], we call our approach the Element Frequency Inverse Phase Frequency weighting (EFIPF). We are also aware that thresholds and other parameter settings need to be investigated, not only for the content prioritzer component but for the entire framework. But again, this work is recent and further studies are needed.

3. PERLIMINARY RESULTS

At the moment, we are evaluating our techniques in a number of ongoing experiments. In [16], we applied our phase detection algorithm to large traces generated from two object-oriented systems: JHotDraw 5.2 and ArgoUML 0.27.

Our phase detection technique was applied to a method call trace generated from ArgoUML by exercising the following scenario: Starting up ArgoUML, drawing a class on the class diagram, and quitting ArgoUML). The resulting trace contained 35754 method calls (to 2331 different methods). Note that a method invocation requires at least two events to be collected, the entry and exit of a method. The trace size in terms of events is therefore about 71508 events, which is considered a relatively large trace. Figure 2 (right upper diagram) shows the result. A clear division of the execution trace into five major phases was also supported by the BIC score with K = 5 as the best fit. As expected, the first phase indicates the initialization of ArgoUML. The second detected phase is concerned with loading auxiliary modules from the input stream and adding them to the Post Load Actions list, which contains actions that are run after ArgoUML has started. The third phase is the phase where the actual class element is drawn. This phase is followed with two other small phases. The first of these phases (i.e., Phase 4) refreshes and updates the models and the last phase (Phase 5) terminates the application. We further applied our technique, with a lower threshold to Phase 3 (drawing a class) to understand how this is accomplished (Figure 2 right bottom). We also applied our content prioritizing technique to find important tasks performed in the trace to summarize it and assign description to each phase. Table 1 shows a task summary for sub-phases of phase 3. The work on prioritizing trace content and generating high-level summaries has not been evaluated yet.

We also applied our smart guided sampling technique to traces of JHotDraw, we found that our approach gave better results in more than 80% of the cases and in all these cases it leads to a more representative sample trace (after manually analyzing each sample). Furthermore, as the size of the sample decreases (to up to 1% of original size) our technique maintains its representativeness while random sampling leads to cases that are significantly unrepresentative of the original trace.

Table 1. Summary of the tasks performed in Phase 3

- Phase 3: Adding a class:
  - Sub-phase 3.1:
    - ⇒ Command to create nodes with the appropriate `modelelement`: Delegate creation of the node to the uml model subsystem and return an object which represents a UML class diagram.
    - ⇒ Define a renderer object for UML Class Diagrams: Return a Fig that can be used to represent the given node.
  - Sub-phase 3.2:
    - ⇒ Prepare the box coordinates to display graphics for a UML Class in a diagram.
    - ⇒ Determine whether the `graphmodel` will allow adding the node (Define a bridge between the UML meta-model representation of the design and the `GraphModel` interface used by GEF).
    - ⇒ Determine if the given object is present as a node in the graph
    - ⇒ Final call at creation time of the Fig, i.e. here the node icon is put on a Diagram: the displayed diagram icons for UML `ModelElements` looks like nodes, has editable names, and can be resized.
    - ⇒ Add the given node to the graph, if of the correct type.
  - Sub-phase 3.3:
    - ⇒ Give continuous feedback to aid in the making of good design decisions: Perform critiques about well-formedness of the model.
    - ⇒ Change the mode of multieditorpane (particularly the `TabDiagrams`) to deselect all tools in the toolbar (Unselect all the toolbar class button).
  - Sub-phase 3.4:
    - ⇒ Hit the class (prepare selection of the class diagram).
    - ⇒ Compute handle selection, if any, from cursor location.
    - ⇒ Prepare selection of the current element.

4. CONCLUSION

In this paper, we drew parallels between trace analysis and the human perception system. We have developed a trace analysis framework inspired by the way the human brain perceives regions and shapes.

The studies presented in this paper are preliminary. Some experimentation has been conducted as proof of concepts. The results are promising.

5. REFERENCES

[1] Uchida, N. et al. 2006. Seeing at a glance, smelling in a whiff: rapid forms of perceptual decision making. *Nature. Rev. Neurosci.* 7, 485–49.

[2] Miller, G. A. 1956. The magical number seven or minus two: Some limits on our capacity of processing information," *Psychol. Rev.*, vol. 63, 81–97.

[3] Frintrop, S., Rome, E., Christensen, H. 2010. Computational visual attention systems and their cognitive foundation: a survey, *ACM Trans. on Applied Perception.* 7.

[4] Wolfe, J.M., Cave, K.R. 1999. The psychophysical evidence for a binding problem in human vision. *Neuron* 24, 11–17.

[5] Boothe, R. G. 2002. Perception of the Visual Environment. *Springer-Verlag*, New York, 13.

[6] Salton G., Buckley C. 1988. Term-weighting approaches in automatic text retrieval. *Inform. Process. Manag.*, 24(5).

[7] Cochran, W. G. 1977. Sampling Techniques. *John Wiley & Sons, Inc.*, New York, NY.

[8] Grabowecky, et al. 1993. Preattentive processes guide visual search. *Journal of Cognitive Neuroscience*, 5, 288-302.

[9] Treisman A.M., Gelade, G. 1980 "A Feature-Integration Theory of Attention," *Cognitive Psychology*, 12, 97–136.

[10] Dugerdil Ph. 2007. Using trace sampling techniques to identify dynamic clusters of classes. *Proc. of CASCON.*

[11] Ball T. 1999. The Concept of Dynamic Analysis. *Proc. 7th European Software Engineering Conference (ESEC'99).*

[12] Reiss, S. P. 2007. Visual representations of executing programs, *Jour. of Visual Languages and Computing*, 18, 2

[13] Smith-Gratto, K., and Fisher, M. 1999. Gestalt theory: A foundation for instructional screen design, *Journal of Instructional Technology Systems*, 27(4), 361–371

[14] Chan A., Holmes R., Murphy G.C., Ying A.T.T.l. 2003. Scaling an Object-oriented System Execution Visualizer through Sampling, *in Proc. of the IWPC03*, 237-244.

[15] Brunk, H.D. 1960. An introduction to mathematical statistics, *Ginn and Company*, Boston.

[16] Pirzadeh, H., Hamou-Lhadj, A. 2011. A Novel Approach Based on Gestalt Psychology for Abstracting the Content of Large Execution Traces for Program Comprehension, *to appear in Proceedings of ICECCS '11.*