2011 International Conference on Software and Computer Applications
IPCSIT vol.9 (2011) © (2011) IACSIT Press, Singapore

A Mind Map Based Framework for Automated Software Log File Analysis

Dileepa Jayathilake [1]

Department of Electrical Engineering
University of Moratuwa, Sri Lanka.

Abstract. Software log file analysis is involved heavily in both Software development and maintenance phases. It serves for various purposes such as verifying the conformance of the software functionality to the specification, software quality check and troubleshooting. Application log files or the logs generated by other monitoring tools are subjected to analysis for extracting information that can be vital in an investigation. These tasks demand expertise to a great deal and are labor intensive when performed manually. The lack of a commonly used technique to record expert knowledge stands as an impediment to automate the analysis tasks. The need for correlating information extracted from different locations in the same log file or multiple log files further ads to this complexity. This paper describes a framework based on mind maps which formulates a homogeneous platform for recording expert knowledge as well as for performing other tasks such as extracting information from log files, drawing inferences and creating reports. The framework includes a scripting language, a parallel application programming interface and a set of tools. Usage is illustrated by a proof of concept system built using the framework that creates a useful report after analyzing a log file generated by a widely used software monitoring tool.

Keywords: log file analysis, correlating log data, mind maps, software quality verification, troubleshooting, expert system, knowledge representation, test automation, software analysis

1. Introduction

Automated software log file analysis is a discipline which received relatively less attention in the industry. On the other hand, its importance has become paramount given that the maintenance cost of software exceeds the initial development cost for almost all the long running software projects. Bugs surfacing during later stages are proven to incur significant costs to software vendors. This creates the need for proactive early detection of bugs which requires a greater focus on software functional conformance and quality verification. Furthermore, efficient troubleshooting plays a significant role in reduction of maintenance cost. All these processes generally include log file analysis to a great extent [1]. Typically involved in an analysis are the application log files that contain information regarding important application events and the logs generated by monitoring tools that report on certain aspects of the application. For example, one tool can monitor the memory usage of an application while another reports on failed low level API calls. Analysis steps include interpreting the log files to extract information required for the analysis, making inferences and presenting results to be consumed by various stakeholders such as managers, technical leads, developers and quality assurance engineers for further investigation or making decisions. All these steps, when performed manually, consume a large amount of expert's time which results in huge costs.

[1] Dileepa Jayathilake. Tel.: +94 776 985 444; fax: +94 114 721 198
E-mail address: dilj220@gmail.com

It is highly likely that certain recurring patterns in software verification and troubleshooting emerge with the maturity of a product. Limiting this knowledge to a few product experts without proper recording can cause at least two problems to an organization. There is the risk of losing the knowledge asset when an expert leaves the organization. Added to that is the difficulty to automate the recurrence. The lack of a platform to easily and unambiguously record the expert knowledge stands as a barrier for overcoming these problems.

The framework described in this paper is an approach to provide a unified infrastructure for building expert systems that automate each step involved in log file analysis. In addition it formulates means for recording the knowledge of individual experts which can be later aggregated to build an aggregated knowledgebase. The framework supports knowledge representation in the form of mind maps and provides a simple yet powerful scripting language. An application programming interface is also included for greater flexibility. This programming infrastructure is accompanied by a set of tailor-made tools to assist in the analysis steps.

Section 2 provides a detailed identification of the problem and Section 3 describes the goals of the framework in providing a solution. After that the fundamental concepts of the framework are explained in Section 4 which is followed by an overview of the implementation in Section 5. Section 6 illustrates an example usage scenario for the framework. In Section 7, an experiment done as a proof of concept for the framework usage is detailed. Related works are mentioned in Section 8. Section 9 concludes the work and the potential enhancements are discussed in Section 10.

2. Problem Identification

This section provides an insight into the tasks associated with each step in log file analysis and problems and challenges encountered when performing them manually.

2.1. Log file interpretation

Application and tool logs typically grow into huge sizes holding a lot of information that spread through many areas [1]. However, in most cases, a bulk of this information contains normal operational data which is of less importance in an analysis. Extracting the useful information from logs needed for the target analysis is a challenging and a laborious task. Furthermore, analysis procedures often demand extracting information from more than one log file and correlating them to have a broader understanding of the case.

Reading logs usually require tool (or application) expertise. Being a labor intensive and repetitive task, it opens the door for human errors causing incorrect interpretations which may result in erroneous conclusions.

2.2. Making inferences on interpreted information

A lot of knowledge and experience is typically required to infer facts from extracted log data for making conclusions. Failure to make good inferences may delay the analysis in best case and will hinder it from generating any solution to the problem in worst.

2.3. Presenting facts to different levels of stakeholders

A project manager may be interested on the overall health and performance of an application whereas a developer may want information on unusual cases for further analysis. A quality assurance engineer may look for areas that need to be focused on future testing. However, most tools available today do not possess this degree of versatility.

2.4. Handling recurring patterns

Recurring patterns identified in analysis are typically known only to a group of experts and are practised repetitively in a manual fashion. Although this knowledge is sometimes documented in the form of sequential text, it is still far from automation due to the inherent ambiguity in natural language.

3. Framework Goals

Clearly understanding the problems to solve, we come up with a set of goals for the framework. Since the log files generated by most existing applications and tools can easily grow up to several gigabytes during

a typical analysis, one target for the framework is to provide the capability to handle huge log files under typical resource constraints. Flexibility is of paramount importance to deal with widely different problems and user expectations. The framework should promote reuse from its architecture to facilitate expert knowledge dissemination. It should also be scalable so that the users can develop and plug in new modules that serve specific needs. Furthermore, the framework should be simple and be aligned with standard practises so that a new user can start using it with a short learning curve. The knowledge representation mechanism should be easily understood by both engineers and computers to guarantee that the expert knowledge, once recorded, acts as a means of efficient knowledge transfer mechanism as well as a resource for automation. The syntax used should be configurable wherever possible for better usability and to support localization.

4. Framework Concepts

Mind maps resemble the organization of concepts in human brain better than many other representations do [2]. A mind map can be implemented as a tree which is a data structure widely used in programming. Therefore mind maps are selected as the knowledge representation form in the framework. Familiar data types found in popular computer languages are used in the framework in order to shorten the learning curve. Since most tasks involved with automated log file analysis take the form of sequential execution of actions on specific data structures, the framework is equipped with a simple scripting language. Engineers can write scripts to perform tasks such as interpreting log files, inferring facts and generating reports. A recurring analysis pattern can be coded as an isolated section within a script which we call a 'scriptlet'. This scheme allows engineers to automate diverse tasks associated with log file analysis in a unified way while providing the domain experts a mechanism to record their knowledge on a specific functionality as a scriptlet without worrying about the remainder of the script.

5. Implementation

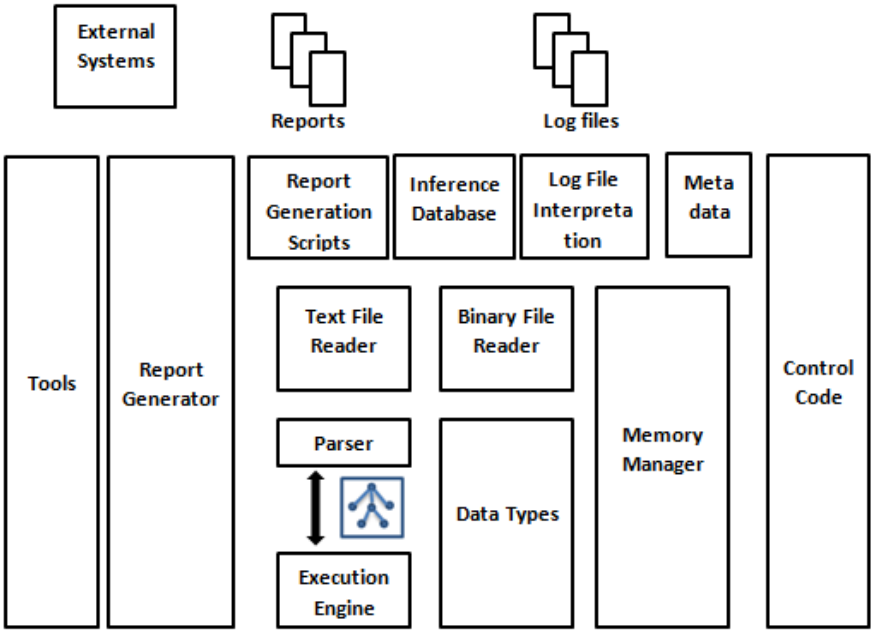

Fig. 1: System Architecture

Fig. 1 shows the system architecture. The framework exposes the basic data types Node, Integer, Float, String, Boolean and Null along with collection types in the form of lists. A list may contain values of basic data types (possibly different types) or other lists as its elements. Every type resembles the general notion of that type in common programming languages. Each type (including collection types) is regarded as a specialization of an abstract type called Entity. This is done to make variables and values of different types accessible in a unified way. Each basic type provides a set of functions to operate on it. In addition, the user can define new functions inside the control code (described later). Every function accepts a parameter of type Entity and outputs another Entity. This scheme not only provides the ability to chain functions seamlessly

inside a single line of code but also enables users to integrate custom functions that operate on an arbitrary input-output combination. A special set of list functions are implemented to facilitate mind map operations.

The scripting language is implemented based on a context free grammar. It is "Turing Complete" [3] so that the entire spectrum of general programming tasks is possible within it. Features such as the ability to add comments and line continuations are supported for better usability. The complete language syntax and the function names are made configurable due to the reasons mentioned in section 3. Configurable syntax and function names are kept as metadata in a single file. A separate metadata file is associated with each script. The execution engine is the run time part of the framework where the actions specified in the scripts are actually executed. The value of each variable at each point in execution and the types of the variables defined inside the script are resolved dynamically in this stage. The actions are performed sequentially in accordance with the control logic.

An application programming Interface (API) is provided in C++ to supplement the scripting mechanism. The API is heavily based on object oriented programming concepts. It exposes a hierarchy of objects to the application programmer. Most operations are duplicated between the scripting language and the API to provide better flexibility when building automated systems.

The framework allocates room for convenience tools inside its architecture. A tool that logs important test events along with a timestamp is included with the current implementation. This tool can be used by quality assurance engineers to capture timing information of different points of a test. File readers are provided to read files in various formats including ASCII text files, UNICODE text files and binary files. Large files are read in an optimum way to keep the memory footprint in a manageable range.

6. Usage

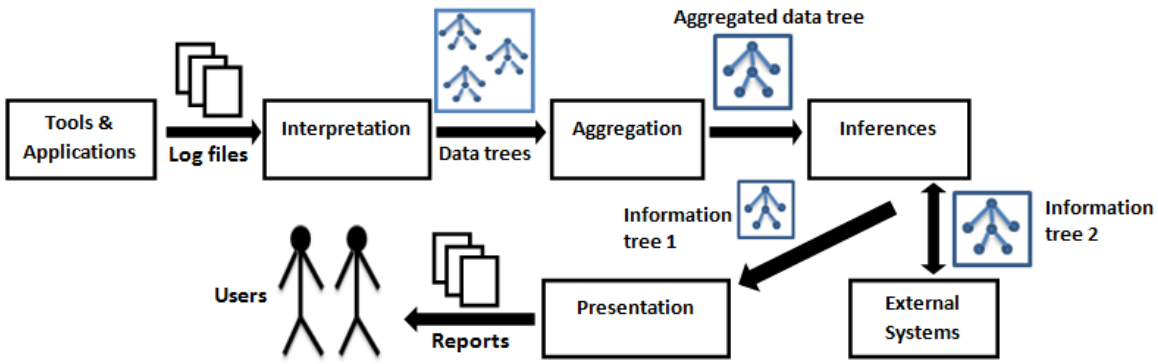

Fig. 2: Example Usage Scenario

Fig. 2 depicts a typical usage scenario for the framework. The analysis starts with a set of log files probably generated by various tools focusing on the same application. A script is written to extract data from each log file. Execution of each script yields a tree containing the data extracted from the corresponding log file. After that another script is used to aggregate the information in these trees into a single information tree. An inference script is applied subsequently on this information tree to get a resulting tree which holds the facts that can be used in further analysis or making decisions. Finally another set of scripts are applied in parallel on this tree to generate reports targeting different levels of people. Each report will contain the information of interest to the consumers of that report. An information tree can also be passed to external systems for further processing.

7. Results

A proof of concept system is developed using the framework to analyze two log files and generate a report. One log file is generated by Application Verifier, which is a free tool from Microsoft for analyzing low level system call failures by an application. The log file is an xml. The second log is taken from the timestamp recording tool provided by the framework. This log is a plain text file. The two tools are simultaneously used to monitor an xml reader application. Few operations are performed in the application and the start and end times of each operation are marked using the recording tool. Data from the resulting

two log files are extracted using scripts and are correlated based on timestamps. Another script is used to generate an html report which summarizes the failures in system calls for each operation in the application and presents them in a table view.

8. Related Work

Automated log file analysis has not been given much attention either in research or in industry. Andrews [4] describes a formal framework for log file analysis which includes a language for specifying analyzer programs. The framework focuses on software functional verification using log file analysis. Eick, Nelson and Schmidt [5] present a tool that graphically displays important information in log files and filters log file noise. This tool provides solutions for extracting significant error messages from cluttered log files, handling unordered messages and finding root cause from cascaded error messages. In the work published by Saneifar, Bonniol, Laurent and Poncelet [6], [7], the heterogeneous and evolving structure of certain log files is analyzed and a method called EXTERLOG for extracting terminology from such log files is presented. Though all these work address interesting domains in log file analysis, they focus on application log files only. Correlating information from multiple log files and report generation are not addressed in any of these work.

9. Conclusions

This paper demonstrated the abilities of a mind map based framework for building expert systems that automate software log file analysis. To our knowledge, it is the first approach for log file analysis using mind maps. It provides the ability to correlate information from multiple log files to derive facts; an area that is not addressed by any previous work. In addition it provides a unified infrastructure for automating a diverse set of tasks associated with log file analysis which would otherwise be labor intensive and error prone. Custom reports can be generated using the same set of tools and procedures used for extracting information and making inferences. The results prove that the collection of utilities bundled with the framework can be driven towards building powerful automated analysis systems that carry out tasks that are either painstaking or impossible if done manually. Veteran software engineers and domain experts can use the mind map based knowledge representation scheme to construct a rich knowledgebase that can be consumed by both human analysts and automated analysis systems.

Requirement for prior knowledge on exact log file structure can pose challenges particularly in cases where the log file structure evolves rapidly. Difficulty to handle unordered log messages is another limitation.

10. Future Work

A script library can be authored targeting widely used software analysis tools to provide out of the box functionality so that the framework can be readily used by a new user even without writing a script. A fuzzy logic module can be plugged in to the framework so that the inferences are expressible in vaguer terms. Improving the scripting language so that the scripting job is more in line with a mind mapping experience will open the door for more non-technical users and other domains such as software specifications, test cases and user experience.

11. References

[1] J. Valdman. Log file analysis. Technical Report DCSE/TR-2001-04, Department of Computer Science and Engineering (FAV UWB), 2001.

[2] Tony Buzan. *The* Mind *Map Book.* Penguin Books, 1996, ch. 2

[3] John E. Hopcroft, Jeffery D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979, pp. 13-137

[4] J. H. Andrews. Theory and practice of log file analysis. Technical Report 524, Department of Computer Science, University of Western Ontario, May 1998.

[5] S. G. Eick, M. C. Nelson, J. D. Schmidt. Graphical Analysis of Computer Log Files. *Communications of the ACM*, Vol. 37, No. 12, pp. 50-56, 1994.

[6] H. Saneifar, S. Bonniol, A. Laurent, P. Poncelet. Mining for relevant terms from log files. In: KDIR'09. *Proc. of International Conference on Knowledge Discovery and Information Retrieval*. Madeira, Portugal. 2009.

[7] H. Saneifar, S. Bonniol, A. Laurent, P. Poncelet. Terminology extraction from log files. In: KDIR'09. *Proc. Of 20th International Conference on Database and Expert Systems Applications*. pp. 769-776. Lecture Notes in Computer Science, Springer 2009.