

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**  
**Jnana Sangama, Belgaum-590 014**



**2013 - 2014**

**A Dissertation Report on**

***“A Knowledge and a Rule Based Engine to  
Analyze the Logs for Troubleshooting”***

*Submitted in partial fulfillment of the requirements for the award of degree  
of*

**MASTER OF TECHNOLOGY**  
*in*  
**INFORMATION TECHNOLOGY**

*by*

**Prashant Achari**  
**(USN: 1RV12SIT10)**

*Under the guidance  
of*

**Dr. Jitendranath Mungara**  
**Dean, PG-Studies,**  
**Department of ISE, R.V.C.E.,**  
**Bangalore - 560059**

**Susanta Adhikary**  
**Architect,**  
**HP India Software Operations,**  
**Bangalore - 560048**



**R.V. COLLEGE OF ENGINEERING,**  
**DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING**  
**Bangalore – 560059**

# VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM

## **R.V. COLLEGE OF ENGINEERING, DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING** Mysore Road, R.V.Vidyaniketan Post, Bangalore - 560059



### **CERTIFICATE**

Certified that the project work entitled “ **A Knowledge and a Rule Based Engine to Analyze the Logs for Troubleshooting**” carried out by **Mr. Prashant Achari**, USN: **1RV12SIT10**, a bonafide student of **R.V. College of Engineering, Bangalore** in partial fulfillment for the award of **Master of Technology in Information Technology** of the **Visvesvaraya Technological University, Belgaum** during the year 2013-2014. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirement in respect of project work prescribed for the said degree.

**Dr. Jitendranath Mungara**  
Dean, PG-Studies,  
Department of ISE, R.V.C.E.,  
Bangalore - 560059

**Dr. N K Cauvery**  
Head of Department,  
Department of ISE,  
R.V.C.E., Bangalore -59

**Dr. B. S. Satyanarayana**  
Principal,  
R.V.C.E.,  
Bangalore -59

**Name of the Examiners**

1. \_\_\_\_\_

2. \_\_\_\_\_

**Signature with Date**

\_\_\_\_\_

\_\_\_\_\_

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM**

**R.V. COLLEGE OF ENGINEERING,  
DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING  
Mysore Road, R.V.Vidyaniketan Post, Bangalore - 560059**

## **DECLARATION**

I, **Prashant Achari**, student of fourth semester M.Tech, in the Department of Information Science and Engineering, R.V. College of Engineering, Bangalore declare that the project entitled “**A Knowledge and a Rule Based Engine to Analyze the Logs for Troubleshooting**” has been carried out by me and submitted in partial fulfillment of the course requirements for the award of degree in **Master of Technology in Information Technology** of **Visvesvaraya Technological University, Belgaum** during the academic year **2013-2014**. The matter embodied in this report has not been submitted to any other university or institution for the award of any other degree or diploma.

**Prashant Dinu Achari**

USN: 1RV12SIT10

Department of Information Science and Engineering,

R.V. College of Engineering,

Bangalore-560059



Hewlett-Packard  
India Software Operation Pvt. Ltd.  
Sy. No. 192, Whitefield Road  
Mahadevapura Post,  
Bangalore - 560 048.  
India  
<http://www.hpindia.com/iso>

**May 29, 2014**

**Bangalore**

**Internship Certificate**

This is to certify that **Prashant Dinu Achari** is working on the **A Knowledge and a Rule Based Engine to Analyze the Logs for Troubleshooting** as a part of his internship and its duration is from July 18, 2013 and July 18, 2014 under the guidance and supervision of **Susanta Adhikary**.

“He is showing great enthusiasm and interest in the project. His understanding of the concept is good and he has effectively participated in the phases of the project. Till date his conduct is excellent”

**Chirag Bhayani**

**Staffing Manager**

**Graduate Hiring**

## **ACKNOWLEDGEMENT**

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance and cooperation of intellectual. A number of personalities, in their own capacities have helped me in carrying out this project work. I would like to take this opportunity to thank them all.

I would like to thank my guide **Dr. Jitendranath Mungara**, Dean, PG-Studies (CSE-ISE), Department of ISE, R.V.C.E, Bangalore, for his guidance, encouragement and assistance throughout this project.

I would like to thank professor and Head of Department **Dr. Cauvery N K**, Department of Information Science and Engineering, RVCE for providing all the support and facility.

I also extend my cordial thank to **Hewlett-Packard India Software Operation Pvt. Ltd.** Bangalore, for providing me an opportunity to carry out the internship in organization. I would like to thank **Mrs. Shalini Agarwal**, Manager, Mr. **Susanta Adhikary**, Mentor, for their support and guidance.

I deeply express my sincere gratitude to **Dr. B. S. Satyanarayana**, Principal, R.V.C.E, Bangalore, for his moral support towards completing my project work.

I thank my Parents, and all the Faculty members of Department of Information Science & Engineering for their constant support and encouragement.

Last, but not the least, I would like to thank my peers and friends who provided me with valuable suggestions to improve my project.

## **ABSTRACT**

The project titled “Knowledge and rule based engine to analyze logs for troubleshooting” is used for efficient log analysis by automating the reparative tasks in the log analysis process and predicting the issues in the logs by the previously known knowledge. Technical support for software or hardware is always a challenge, both from the complexity and investment standpoint. Today the dependency on computers, networked systems and managing applications are becoming more dominant and only log files are available to trace the applications.

Log file analysis process is heavily involved, in software, hardware as well as in network related domains. 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. **There is no technique available to record expert knowledge and this stands as an obstacle to** automate the analysis tasks. Hence there is a need for correlating information extracted from different locations in the same log file or multiple log files further adds to this complexity.

This work describes a practical approach for analysis of the logs using **supervised learning like a decision tree and random forest to classify and predict the issue and to recommend steps for troubleshooting the issue depending on the classification.** The learned pattern is represented in the form of rules and a rule engine based system is used to validate the log files. The overall solution proposed here is to automate the analysis of logs and provide recommendations for faster response to reported problems. The log analyzer self-learns the new issue and provides necessary recommendations and **reduces the redundant data by 82% in analysis.**

# TABLE OF CONTENTS

|  |   |
|--|---|
| ACKNOWLEDGEMENT  | 1 |
| ABSTRACT   | 2 |
| TABLE OF CONTENTS  | 3 |
| GLOSSARY   | 9 |
| CHAPTER 1  | 1 |
| INTRODUCTION TO KNOWLEDGE AND A RULE BASED ENGINE TO<br>ANALYZE THE LOGS FOR TROUBLESHOOTING                       | 1 |
| 1.1 Definitions  | 2 |
| 1.2 State of art   | 3 |
| 1.3 Motivation   | 5 |
| 1.4 Problem Statement  | 5 |
| 1.5 Objective of the Project   | 6 |
| 1.6 Scope  | 6 |
| 1.7 Methodology  | 6 |
| 1.8 Organization of the Report   | 7 |
| 1.9 Chapter Summary  | 7 |
| CHAPTER 2  | 8 |
| SYSTEM REQUIREMENTS SPECIFICATION FOR KNOWLEDGE AND A RULE<br>BASED ENGINE TO ANALYZE THE LOGS FOR TROUBLESHOOTING | 8 |
| 2.1 Overall Description  | 8 |
| 2.2 Specific Requirements  | 9 |

|   |    |
|---|----|
| CHAPTER 3   | 12 |
| HIGH LEVEL DESIGN FOR KNOWLEDGE AND A RULE BASED ENGINE TO ANALYZE THE LOGS FOR TROUBLESHOOTING | 12 |
| 3.1 Introduction  | 12 |
| 3.2 Design Considerations   | 12 |
| 3.3 Assumptions and Dependencies:   | 13 |
| 3.4 Development Methods   | 13 |
| 3.5 System Architecture:  | 14 |
| 3.6 Data Flow Diagram   | 20 |
| 3.7 Sequence Diagram for Log Analyzer   | 22 |
| CHAPTER 4   | 25 |
| DETAILED DESIGN FOR KNOWLEDGE AND A RULE BASED ENGINE TO ANALYZE THE LOGS FOR TROUBLESHOOTING   | 25 |
| 4.1 Structure Chart   | 25 |
| 4.2 Functional Description of the Modules   | 25 |
| CHAPTER 5   | 33 |
| IMPLEMENTATION OF KNOWLEDGE AND A RULE BASED ENGINE TO ANALYZE THE LOGS FOR TROUBLESHOOTING     | 33 |
| 5.1 Programming Language Selection  | 33 |
| 5.2 Platform Selection  | 37 |
| 5.3 Code Conventions  | 37 |
| 5.4 Programming Techniques  | 43 |
| 5.5 Difficulties Encountered and Strategies Used to Tackle                                      | 46 |



|   |    |
|---|----|
| CHAPTER 6   | 47 |
| SOFTWARE TESTING FOR KNOWLEDGE AND A RULE BASED ENGINE TO<br>ANALYZE THE LOGS FOR TROUBLESHOOTING | 47 |
| 6.1 Test Strategy   | 47 |
| 6.2 Levels of Testing   | 48 |
| CHAPTER 7   | 56 |
| RESULTS OF KNOWLEDGE AND A RULE BASED ENGINE TO ANALYZE THE<br>LOGS FOR TROUBLESHOOTING           | 56 |
| 7.1 Evaluation Metrics  | 56 |
| 7.2 Experimental Dataset  | 57 |
| 7.3 Performance Analysis  | 59 |
| 7.4 Output Result   | 61 |
| 7.5 Inferences  | 61 |
| CHAPTER 8   | 62 |
| CONCLUSIONOF KNOWLEDGE AND A RULE BASED ENGINE TO ANALYZE<br>THE LOGS FOR TROUBLESHOOTING         | 62 |
| 8.1 Limitation of the Project   | 62 |
| 8.2 Future Enhancements   | 62 |
| REFERENCES  | 63 |
| APPENDIX  | 70 |
| APPENDIX-A  | 70 |
| User Authentication   | 70 |
| Dash board  | 70 |
| Upload the details  | 72 |

|                           |    |
|---------------------------|----|
| Running the Analysis      | 72 |
| APPENDIX-B                | 74 |
| Technical Paper published | 74 |

## LIST OF FIGURES

|  |    |
|--|----|
| Figure 3.1 Log Analyzer System Architecture                    | 15 |
| Figure 3.2 Data Formatter Elements                             | 16 |
| Figure 3.3 Rule Engine Elements                                | 17 |
| Figure 3.4 Analytics Engine Elements                           | 19 |
| Figure 3.5 Level 0 DFD for Log Analyzer System                 | 20 |
| Figure 3.6 Level 1 DFD for Log Analyzer System                 | 21 |
| Figure 3.7 Level 1 DFD for Log Analyzer System                 | 22 |
| Figure 3.8 Sequence Diagram for Log Analyzer                   | 23 |
| Figure 4.1 Flow Chart for Data Formatter                       | 27 |
| Figure 4.2 Flow Chart for Rule Engine Backward Chaining        | 29 |
| Figure 4.3 Flow Chart for Analytics Engine                     | 31 |
| Figure 7.1 Log Analyzer CPU utilization for 1.54GB log dump    | 60 |
| Figure 7.2 Log Analyzer Memory utilization for 1.54GB log dump | 60 |
| Figure 8.3 Log Analyzer CPU utilization for 1.54GB log dump    | 61 |
| Figure A.1 : Login Module for the log analyzer.                | 70 |
| Figure A.2 : Dash board for the log analyzer                   | 71 |
| Figure A.3 : Dash board for the log analyzer with file browser | 71 |
| Figure A.4 : Uploading the details for the log analyzer        | 72 |
| Figure A.6 :Analysis result with recommendation.               | 73 |

## LIST OF TABELS

|   |    |
|---|----|
| Table 6.1: Unit Test Case 1 for File Uploading                                      | 49 |
| Table 6.2: Unit Test Case 2 for File Extraction                                     | 49 |
| Table 6.3: Unit Test Case 3 for File Browser and Reader                             | 50 |
| Table 6.4: Unit Test Case 4 for Lexical Analyzer                                    | 51 |
| Table 6.5: Unit Test Case 5 for Rule Engine   | 51 |
| Table 6.6: Unit Test Case 5 for Analytic Engine                                     | 52 |
| Table 6.7: Integration Test Case 7 for File uploading extracting and browsing       | 53 |
| Table 6.8: Integration Test Case 7 for formatter, rule engine and analytic engine   | 54 |
| Table 6.9: Integration Test Case 7 for rule engine, analytic engine and recommender | 54 |
| Table 8.1: Training data  | 57 |
| Table 8.8: Classification report  | 58 |

# **GLOSSARY**

**DFD** : Data Flow Diagram

**GUI** : Graphical User Interface

**OS** : Operating System

**SRS** : Software Requirement Specification

**CPU** : Central Processing Unit

**REST** : Representational state transfer

**SOAP** : Simple Object Access Protocol

**IG** : Information Gain

**WA** : Weighted Average

**API**: Application Program Interface

**RML** : Rule Markup Language

## CHAPTER 1

# INTRODUCTION TO KNOWLEDGE AND A RULE BASED ENGINE TO ANALYZE THE LOGS FOR TROUBLESHOOTING

Software and hardware log files are analyzed for many purposes [1]. In certain cases, the goal is to **verify the functional conformance of** the software with a given specification where the log file entries are observed to confirm that the application generates desired outputs at intended instances [2]. Log files are heavily used in **software troubleshooting**. When an application malfunctions in production, the only trace available for developers to investigate the cause, more often than not, is the application log file. In addition, log file analysis helps in extracting **vital usage statistics** from web servers and in **security monitoring** [3]. In security it used in intrusion detection system by detecting the irregular pattern in the log file. Potential uses of log file analysis extend beyond the popular practices mentioned above. Many powerful software analysis tools have been emerging over the recent past with capabilities to monitor software with respect to memory leaks, corruptions, IO overhead, access failures, performance bottlenecks, security pitfalls, failed lower level API calls, etc. [4]. Almost every tool generates some form of a log which can be used to identify software defects that can be hardly detected in conventional testing. **Correlating data extracted from a monitoring tool with an application log can reveal valuable information on system level events caused by important application events** [5].

Logs are emitted by network devices, operating systems, applications and all manner of intelligent or programmable device. A stream of messages in time-sequence often comprises a log. Logs may be directed to files and stored on disk, or directed as a network stream to a log collector. Logs are often created by software developers in order to aid in the debugging of the operation of an application and to verify the state of the system. The syntax and semantics of data within log messages are usually application or vendor-specific. Terminology may also vary; for example, the authentication of a user to an application may be described as a login, a logon, a user connection or authentication

event. Hence, log analysis must interpret messages within the context of an application, vendor, system or configuration in order to make useful comparisons to messages from different log sources [5].

The dependency on computers, networked systems and managing applications are becoming more dominant [6]. With increasing complexity in design, introduction of new technologies, the issues are not trivial, hence technical support team spend bulk of their time to go over the log file and correlate them to provide the resolution. Despite the benefits of it, software log file analysis is a labor intensive and error prone activity when performed completely manually. Furthermore, it generally demands domain and tool expertise which is an expensive resource [7]. Therefore, automating at least a part of the process is of paramount importance. The first step in automated log analysis is automatic extraction of relevant information from log files. There are many log analyzers available to analyze the hardware or software failure issues, they can address very specific issues and not enough intelligence to learn new log pattern for analyzing an issues and providing recommendations. It is required to have more flexible and predictive approach for troubleshooting and monitoring of the resources to resolve the errors in the system. Here we try to take a practical approach for predictive analysis of the application logs. Overall the solution proposed is to enable the analysis of logs and provide recommendations for faster response to reported problem and also provide a self-learning new issue and provide probable recommendations.

## 1.1 Definitions

Various terminologies related to this project have been explained in this section.

- **Production rule system:** A production rule system is a computer program typically used to provide some form of artificial intelligence, which consists primarily of a set of rules about behavior. These rules, termed productions, are a basic representation found useful in automated planning, expert systems and action selection. A production system provides the mechanism necessary to execute productions in order to achieve some goal for the system.
- **Working memory:** A working memory is the system that actively holds multiple pieces of transitory information in the mind, where they can be manipulated.

- **RIF:** Rule Interchange Format is a W3C Recommendation. RIF is part of the infrastructure for the semantic web.
- **DRL:** A DRL file is a **rule file** where you can have multiple rules, queries and functions, as well as some resource declarations like imports, global and attributes that are assigned and used by your rules and queries. However, you are also able to spread your rules across multiple rule files.
- **Rule Repository:** A database for storing the business rules as defined by the business users.
- **Rule Editor:** An intuitive user interface that allows business users to define, design, document and edit business rules.
- **Reporting Component:** An intuitive user interface that allows business users to query and report existing rules.
- **Rules Engine Execution Core:** The actual programming code that enforces the rules.


## 1.2 State of art

The dependencies on computer, networked system and managing application are becoming more dominant. With increasing complexity in design, introduction of new technologies, the issues are not trivial, hence technical support team spend bulk of their time to go over the log file and correlate them to provide the resolution. Despite the benefits of it, software log file analysis is a labor intensive and error prone activity when performed completely manually [8][9]. Furthermore, it generally demands domain and tool expertise which is an expensive resource. Therefore, automating at least a part of the process is of paramount importance. The first step in automated log analysis is automatic extraction of relevant information from log files. There are many log analyzers available to analyze the hardware or software failure issues, they can address very specific issues and not enough intelligence to learn new log pattern for analyzing an issues and providing recommendations. It is required to have more flexible and predictive approach for troubleshooting and monitoring of the resources to resolve the errors in the system [10].

**This fundamental limitation of the current log analysis system is that the recurring error patterns are not captured and every time the error occurs the user has to start over again on the repeated issue.** This limitation can be overcome by capturing the work of the user in the form of rules, which will be validated against the new arriving logs.



Decision tree learning is used in classifying the logs. The goal is to create a model that predicts the value of a target variable based on several input variables. A decision tree is a simple representation for classifying examples. Decision tree learning is one of the most successful techniques for supervised classification learning. The log files have finite discrete domains, and there is a single target feature called the classification. Each element of the domain of the classification is called a class. A decision tree or a classification tree is a tree in which each internal node is labeled with an input feature. The arcs coming from a node labeled with a feature are labeled with each of the possible values of the feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes [10].

 **Rete algorithm** is the mainstream of the algorithm in the rules engine; it provides efficient local entities data and rules pattern matching method. Rete algorithm applied in rules engine has defects in performance and demand aspect; here three methods to improve the Rete algorithm in the rule engine are used: rule decomposition, Alpha-Node-Hashing and Beta-Node-Indexing. Rules engine is enterprise applications, business logic framework for extracting rules from software application, and make it more flexible. It makes Rule Engine widely meet the application demand and be greatly improved the efficiency [11]. Further the technique to analyze and correlate the different types of computer log files. Log files are generated from servers and network devices to record operations that occur in the computers and networks [10]. As log files are too enormous to manually analyze, a tool to maximize accuracy as well as efficiency while high speed processing is the goal. The accuracy is increased by using learning algorithms to classify the normal operations from the abnormal ones such algorithms include Naïve Bayes, k-means clustering, and decision tree and for less accuracy, in order to gain speed for both with or without parallel processing techniques. Here they also construct an adaptive learning algorithm to update the model. Then we flush out out-of-date model while the logs are being captured and processed. The result can achieve the goal as they can reach about 30-40% in real-time processing with nearly zero false positive results.

### 1.3 Motivation

The motivation is to design a project that has flexible and predictive approach for troubleshooting and monitoring of the resources to resolve the errors in the system. Here we try to take a practical approach for predictive analysis of the application logs. Overall the solution proposed is to enable the analysis of logs and provide recommendations for faster response to reported problem and also provide a self-learning new issue and provide probable recommendations.

As an academic project with motivation of learning and exposing following technologies

- Machine learning
- Rule engine
- Log analysis
- Analytics engine
- Data Structures and Algorithms
- Data mining
- Java programming

### 1.4 Problem Statement

Lack of efficient log analyzer application for software or hardware issue log analysis has been more of manual task [12]. Mostly in legacy applications where the logs are unstructured [13], the analysis of these logs requires going over the logs and finding any relevant pattern based on the reported problem. Especially for the support team the analysis of these logs becomes difficult and providing first level recommendation or troubleshooting becomes difficult. Apparently the support team would need assistance from development team which incurs cost. A log contains error messages, application specific data and operational uses etc. [14][15]. The traditional approaches of manually analyzing activities in the logs are time consuming and error prone. In most legacy systems logs are unstructured and require engineering efforts for analysis. Also the analysis is done when a functional problem is reported. There is no tool which can be used for continuous analysis and captures if there are any repetitive error occurring in the system and provide recommendations before a major breakdown [15] [16].

Here the challenges of analyzing the existing log files using probabilistic analysis to provide aggregated information from the log files and issued alerts and recommendations based on the data present in the logs [17]. The solution is designed with flexibility for future extension of any new generation or system log analysis with scalability in mind to process large log files.

### **1.5 Objective of the Project**

The following are the objectives of the project;

- Automating collection of logs on notification.
- Extract valuable information from unstructured log file and troubleshoot hardware and software related issues.
- Remove irrelevant information thereby reducing the log file size.
- Analytics engine comprise of log specific rule to analyze the logs.
- Provide recommendations based on the analyzed information and knowledgebase.

### **1.6 Scope**

This project will find useful in corporate environment where systems administrator need to analyze the log file manually and troubleshoot the issue. The pattern of troubleshooting the issue is captured by the analytics engine and engineer need not have to go over the log file if the specific issue was already resolved, the analyzer will automatically recommend the possible solution.

### **1.7 Methodology**

Adaptive log analysis process includes

- The collection of logs can be automated based on notification
- Provide data formatter engine to convert the logs into a standard format, remove irrelevant information to reduce size of information to be analyzed.
- Analytic engine which comprise of log specific rule to analyze the logs
- Provide recommendation based on the analyzed information and knowledge base of known pattern.
- Provide way to auto learn a pattern and provide new formatter and also derive new rule.

The data formatter engine will process the unstructured logs from the support dump with the associated formatter [17]. The engine does auto activation of the data processing based on the compute required. The rule engine will assign rules for specific logs would use the knowledge from the knowledge base. Based on the analysis it will provide necessary recommendation available in data store. The data store is distributed to meet the scalability. The knowledge base will be kept updated manually or through feed. The critical component is the learning engine that provides the solution by providing an intelligent way to learning logs pattern and rules required to process the analysis.

## **1.8 Organization of the Report**

The report is organized into following sections:

In chapter 1, Introduction to the subject matter where the concept of Log analysis and knowledge base System is explained. It also talks about the motivation behind the project and gives a brief methodology adopted to achieve the objectives.

In chapter 2, Concepts log analysis, the rule base system and learning algorithms used in the system are explained. Also a brief introduction of the environment is provided.

In chapter 3, the flowchart, block diagram and the specifications associated with the log analyzer are discussed is discussed.

In chapter 4, the implementation of the flowchart discussed in chapter 3 in java using Eclipse IDE is presented along the algorithm for the analysis

In chapter 5, the testing done on the software has been discussed. Three kinds of testing have been done, Unit testing, Integration Testing and System Testing.

In chapter 6, results of the project are presented. The results are shown through graphs.

In chapter 7, the conclusion, limitations of the project and the further work are stated.

## **1.9 Chapter Summary**

This chapter gives brief idea of purpose, scope and motivation of the dissertation. Literature survey in this chapter gives related work carried out earlier. At the end of the chapter problem definition, objectives, methodology and organization of report are narrated.

## **CHAPTER 2**

# **SYSTEM REQUIREMENTS SPECIFICATION FOR KNOWLEDGE AND A RULE BASED ENGINE TO ANALYZE THE LOGS FOR TROUBLESHOOTING**

### **2.1 Overall Description**

A Software Requirements Specification (SRS) is a complete description of the behavior of the system to be developed. It includes functional requirements, non-functional (or supplementary) requirement which imposes on the design or implements performance engineering requirements, quality standards or design constraints.

#### **2.1.1 Product Perspective**

The project should be able to provide flexible and predictive approach for troubleshooting and monitoring of the resources to resolve the errors in the system. The solution proposed is to enable the analysis of logs and provide recommendations for faster response to reported problem and also provide a self-learning new issue and provide probable recommendations.

#### **2.1.2 Product Functions**

- Logs should be collected automatically by notification of error in the system.
- Data formatter engine should convert the logs into standard format and remove the irrelevant information and thereby reduce the file size.
- Analytics engine should comprise of log specific rule to analyze the loges.
- Provide recommendations based on the analyzed information and knowledgebase of known pattern.
- Should provide a way to auto-learn the pattern and provide new formatters and drive new rule.

### **2.1.3 User Characteristics**

- User should be familiar with the manual log analysis.
- The user will interact with the system with the help of web interface.
- User should be able to troubleshoot from the provided recommendation.
- User should be able to provide proper feedback for training the system.

### **2.1.4 General Constraint**

- Even though the system is designed to work in large scale environment, its performance depends on the analytics engine and rule engine.
- The training data available should large enough to construct the rules.

### **2.1.5 Assumptions and Dependencies**

- The analytics engine which is the back bone of the system should have enough training data so that it can predict the proper resolution and will provide the recommendation.
- If the resolution includes the new pattern of steps for troubleshooting, then new pattern will be updated in to the knowledge base by the user.

## **2.2 Specific Requirements**

This section of the SRS contains all the software requirements to a level of detail, sufficient to enable designers to design a system and to satisfy all those requirements.

### **2.2.1 Functionality Requirements**

- Logs should be able collected automatically by notification of error in the system.
- Data formatter engine should be able convert the logs into standard format and remove the irrelevant information and thereby reduce the file size.
- Analytics engine should comprise of log specific rule to analyze the loges.
- Provide recommendations based on the analyzed information and knowledgebase of known pattern.
- Should provide a way to auto-learn the pattern and provide new formatters and drive new rule.

### 2.2.2 Performance Requirements

- Should be able to manage any type and number of log files.
- Should be able to extract the necessary information from the dump of logs.
- Data extraction from the log file should be ubiquitous and should be time efficient.
- Should be able to provide proper and self-healing recommendations.

### 2.2.3 Supportability

- The client system should be able to upload log files.
- The client system should be able decrypt the file if the files are encrypted.
- GUI is designed using jsp hence web server should support hosting java Applications.

### 2.2.4 Software Requirements

- Operating System
  - Windows or Linux with Servlet container Tomcat 6.0 or above.
  - Windows or Linux with Java 1.6 or above.
  - Windows 7 for Development environment.
- Programming Tool:
  - eclipse IDE
  - Drools plugin
  - Maven plugin
- Platform
  - Java 1.6 or above.
- Other Tools/Libraries
  - Drools Camel server
  - Tomcat 7.0
  - Web browser

### 2.2.5 Hardware Requirements

- Intel Pentium 4 Processor(min), 1GB RAM, 80GB HDD
- LAN/Internet Connection to central server.
- HTTP network communication between user and the module.

### **2.2.6 Design Constraints**

The log attribute field information is required in the parsing the log file, the uploaded log file should have associated metadata about the log file. The recommendation provided for the troubleshooting depends on the available training data and the previously identified issue in the system. The system learns by feedback from the user.

### **2.2.7 Interfaces**

Interface is further classified into the following user, communication and software interfaces

#### **2.2.7.1 User Interfaces -**

The external users are the authorized system administrator of the organization. The administrator can have privilege to collect the log and necessary information from the user. And has the access to perform any operation on the target system to troubleshoot the issue. User has privileges to tune the rules by standard interface and update the knowledge base.

#### **2.2.7.2 Communication Interfaces –**

Communication between user and the central server takes place via HTTP protocol to upload the data, and for request for services. The components of the system uses RESTful services to access each function.

#### **2.2.7.3 Software Interfaces –**

The operating systems on target machines should be Windows which supports TCP/IP protocol and java with required libraries. **The rule engine used is the JBoss Drools Camel server which provides decision making as a service to the requesting modules.**



## **CHAPTER 3**

# **HIGH LEVEL DESIGN FOR KNOWLEDGE AND A RULE BASED ENGINE TO ANALYZE THE LOGS FOR TROUBLESHOOTING**

### **3.1 Introduction**

The purpose of the design phase is to plan a solution of the problem specified by the requirement document. This phase is the first step in moving from the problem domain to the solution domain. The design of the system is perhaps the most critical factor affecting the quality of the software. Here we build the system Block diagram that will be helpful to understand the behavior of the system. Here we divide problem into modules. Data flow Diagrams show flow of data between/among modules.

This chapter presents the following

- Design Considerations: This section describes many issues, which need to be addressed or resolved before attempting to device a complete design solution.
- Assumptions and Dependencies: Describe any assumptions or dependencies regarding the software and its use.
- General Constraints
- Development Methods
- Architectural Strategies
- System Architecture

### **3.2 Design Considerations**

Java is powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. It is easy to use and is therefore easy to write, compile, debug, and learn than other programming languages. This allows you to create modular programs and reusable code. **Java provides the plugin for JBoss Drools rule engine.**

Drools is Java based Rule Engine, which allows for support of multiple rule engines from a single API. It does not deal in any way with the rule language itself. Systems also has rule language standard called RML. It should be remembered that the JSR94 standard represents the "least common denominator" in features across rule engines. This means that there is less functionality in the JSR94 API than in the standard Knowledge. So, by using JSR94, you forfeit the advantage of using the full capabilities of the Drools Rule Engine. It is necessary to expose further functionality, like global and support for DRL, DSL and XML, via property maps due to the very basic feature set of JSR94; this introduces non-portable functionality. Furthermore, as JSR94 does not provide a rule language, you are only solving a small fraction of the complexity of switching rule engines with very little gain. So, while we support JSR94, for those that insist on using it, we strongly recommend you program against the Knowledge API.

### 3.3 Assumptions and Dependencies:

The general assumption of this particular project work has to be explained with proper understanding of the functional aspect of this research work. The application is assumed to be used by system administrator in the corporate environment to manage the datacenter in the IT environment is up to date to make the systems less prone to security vulnerabilities. Here the incoming log file is rendered by the Log Data parser, which converts the log in universal format. This helps to maintain the Log Data Store in structured format. The amount of the data to be extracted from the log file is dynamically decided by machine learning. A Decision tree is used to classify the log data and constructed nodes of the tree are decided using the entropy and training data present in the knowledge base. The resulting decision tree is used to derive the rules for rule engine, which will in turn decide the amount of data to be extracted from the log file. There are multiple log specific formatters. The formatter is selected depending on the rules present in another rule file. Once the information is extracted from the log file, it is analyzed by the Analytics engine.

### 3.4 Development Methods

The development method used in this software design is the modular development method. In this, the system is broken down into different modules, with a certain amount of dependency among them. The input-output data that flows from one-module to another will show the dependency.

Data flow diagrams have been used in the modular design of the system. Structure charts have been used to show the hierarchy of the function calls in the system. Data flow diagrams are mentioned.

The system is developed using the Bottom-up approach. The bottom-up approach is the piecing together of systems to give rise to grander systems, thus making the original systems sub-systems of the emergent system. In this approach the individual base elements of them are first specified in great detail. These elements are linked together to form larger subsystems, which in turn are linked, sometimes in many levels, until a complete top-level system is formed. This strategy often resembles a "seed" model, whereby beginning are small but eventually grow in complexity and completeness.

- Java- programming language has been used to develop the modules.
- Developing the application module by module takes the work of the developer as well as tester or analyzer easier.

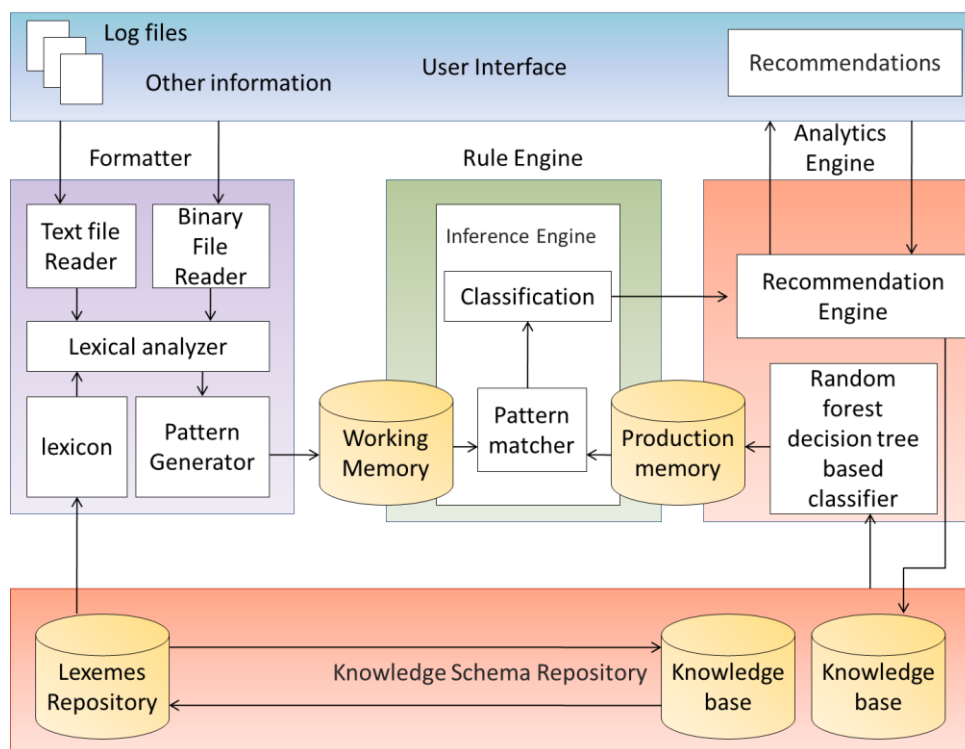
### **3.5 System Architecture:**

The architecture design of this project work is very complex and this is increased at the time of coding process for the development of enabling features of the application. The approach used in the application is object oriented wherein the system is classified into different objects, which represent real world entity.

The system has following components broadly classifying the architecture elements into

- Formatter
- Rule Engine
- Analytics engine
- Graphical User Interface

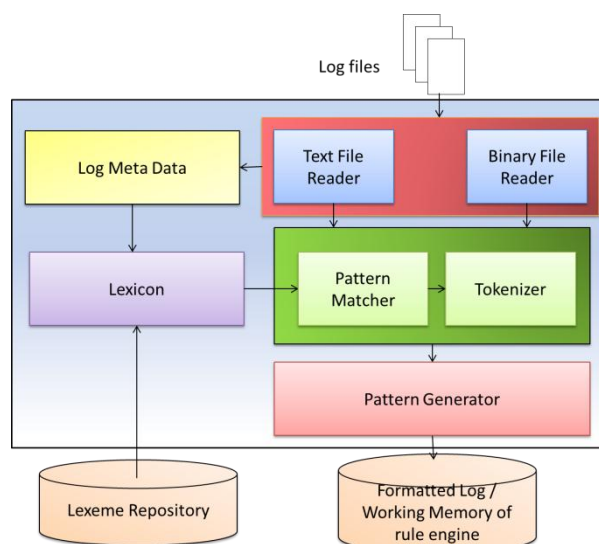
The figure 3.1 shows the overall architecture of the log analyzer.



**Figure 3.1 Log Analyzer System Architecture**

### 3.5.1. Data formatter:

Here the incoming log file is rendered by the Log Data Phraser, which converts the log in standard format. The formatter engine process unstructured logs from the support dump with the associated formatter. The engine the auto activation of the data processing based on the compute required. For support dump like IaaS controller which manages hundreds of thousands of the nodes, the collected support dump is huge. The logs from various sources and platform have different file formats and also there is no standard followed in logging, which make it difficult in analyzing the logs. **Formatter consists of lexical analyzer which searches for specific patterns available in the knowledge base and produces a structured data. The each attribute in the knowledge representation matrix is assigned with specific value from the knowledge base; the attributes are parameters like time stamp, source and destination IP for a network related log and source component, session id and other related information for application log. Regular expression is used to match the pattern. The data formatter engine also updates the new formatter from the learning engine for the new set of application/hardware logs. The formatter lexically analyzes the log file data and extracts the necessary information from the log file. Figure 3.2 shows the elements in the data formatter**



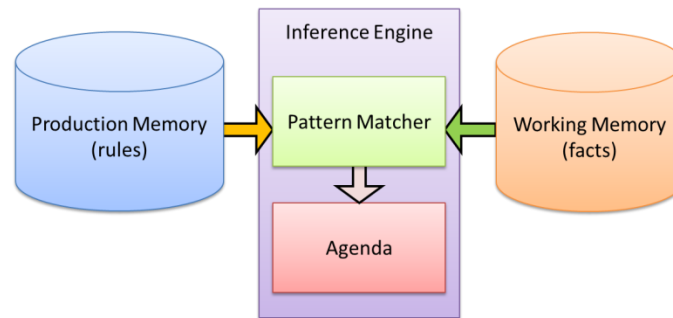
**Figure 3.2 Data Formatter Elements**

The log files from the source are given as an input to the formatter. Formatter is a lexical analyzer. A lexicon is derived from the previously defined knowledge. Each lexicon has n to one relation. The entries in the unstructured log are scanned by either text file reader or a binary file reader depending upon the file format. The log file is searched for specific patterns that are defined in the knowledge schema. Depending on the found pattern, a lexicon is selected belonging to a particular attribute of the knowledge schema. Pattern generator is used to fill up the knowledge schema depending upon the patterns.

### 3.5.2. Rule engine:

This is the backbone of the solution which is the rule engine, which would assign associated rules. The rules for the specific logs would use the knowledge base for searching/ data mining the issue pattern and their resolution. Based on the analysis it would provide necessary recommendation from the knowledge base. The knowledge is typically tagged historical/known issue and their recommendation available in the data store. The data store could be distributed to meet the scalability. For the issue the knowledge base would be kept updated manually or through feed. It would also get update from the learning shared by the customer. The recommendation is the analyzed data. On collected aggregated information, error and pattern recommendation can be created with probable error and cause with the suggestive manual corrective actions to be taken by application/system user. This system uses any new acquired knowledge to feedback within the system and updates the knowledge for suggestive new enhanced rules or data

formatter enhancement for better future analysis. Also would be fed to HP support center to share with the customer. The support services would be exposed as cloud services where customer can register to avail that. Figure 3.1 shows the elements of the rule engine.



**Figure 3.3 Rule Engine Elements**

The rule engine provides following advantages

- **Declarative Programming**

The key advantage of this point is that using rules can make it **easy to express solutions to difficult problems** and consequently have those solutions verified. **Rules are much easier to read than code.**

- **Logic and Data Separation**

The **data is in the domain objects**, the **logic is in the rules**. This is fundamentally breaking the OO coupling of data and logic, which can be an advantage or a disadvantage depending on your point of view. The upshot is that the logic can be much easier to maintain as there are changes in the future, as the logic is all laid out in rules. This can be especially true if the logic is cross-domain or multi-domain logic. Instead of the logic being spread across many domain objects or controllers, it can all be organized in one or more very distinct rules files.

- **Speed and Scalability**

The Rete algorithm, the Leaps algorithm, and their descendants such as Drools' ReteOO, provide very efficient ways of matching rule patterns to your domain object data. These are especially efficient when you have datasets that change in small portions as the rule engine can remember past matches. These algorithms are battle proven.

- **Centralization of Knowledge**

By using rules, you create a repository of knowledge (a knowledge base) which is executable. This means it's a single point of truth, for business policy, for instance. Ideally rules are so readable that they can also serve as documentation.

- **Tool Integration**

Tools such as Eclipse (and in future, Web based user interfaces) provide ways to edit and manage rules and get immediate feedback, validation and content assistance. Auditing and debugging tools are also available.

- **Explanation Facility**

Rule systems effectively provide an "explanation facility" by being able to log the decisions made by the rule engine along with why the decisions were made.

- **Understandable Rules**

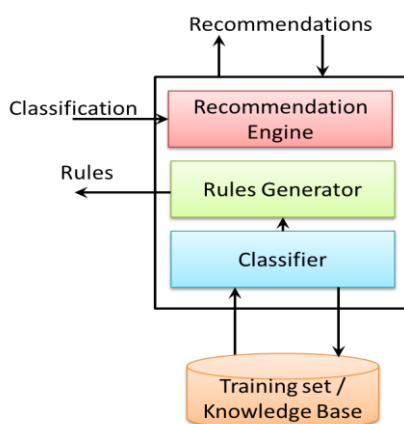
By creating object models and, optionally, Domain Specific Languages that model your problem domain you can set yourself up to write rules that are very close to natural language. They lend themselves to logic that is understandable to, possibly nontechnical, domain experts as they are expressed in their language, with all the program plumbing, the technical know-how being hidden away in the usual code.

Domain experts often possess a wealth of knowledge about business rules and processes. They typically are nontechnical, but can be very logical. Rules can allow them to express the logic in their own terms. Of course, they still have to think critically and be capable of logical thinking. Many people in nontechnical positions do not have training in formal logic, so be careful and work with them, as by codifying business knowledge in rules, you will often expose holes in the way the business rules and processes are currently understood.

As rule engines are dynamic, they are often looked at as a solution to the problem of deploying software. Most IT departments seem to exist for the purpose of preventing software being rolled out. But the rule engines work best when you are able to write declarative rules.

### 3.5.3. Analytics engine:

This is the learning engine the critical component of the solution by providing an intelligent way to learn logs pattern and rules required to process for analysis. The newly learned formatter or rules undergo a training phase to make sure it clearly able to perform the job which would be certified via manual inspection or via automated test engine. Figure 3.4 shows the elements of the analytics engine.



**Figure 3.4 Analytics Engine Elements**

### 3.5.4. User Interface

This is where users interact with our system, the user selects the request by menu driven user interface, the log files from support dump is uploaded through the user through a web based interface. Browse the uploaded files and derive conclusion from the data, filter the data to be analyzed and fed into the analytics engine. This gives web administrators the ability to manage the appearance and features available in the user interface to fit each user's skill level. Its users and groups system allows control over content editing and viewing privileges, and the versioning and workflow systems allow for content approval hierarchies and tracking of content as it moves through the site. Everything in WebGUI is a template which allows for customization, while keeping the site content and style separate.

WebGUI is built as an application framework, and has a pluggable architecture to aid in the extensibility of applications. Developers can create custom applications and functionality that match an organization's processes. A pluggable macro architecture allows for even more extensibility and flexibility. WebGUI's modular design allows code to be easily accessed, changed, and replaced.



### 3.6 Data Flow Diagram

The log analyzer application system can be decomposed into three levels such as level 0, level 1 and level 2. Dataflow models are an intuitive way of showing how data is processed by a system. At the analysis level, they should be used to model the way in which data is processed in the existing system. The notation used in these models represents functional processing, data stores and data movements between different modules.

#### 3.6.1 Data Flow Diagram – Level 0

The level 0 is the initial level Data Flow Diagram and it's generally called as the Context Level Diagram. It is common practice for a designer to draw a context-level DFD first which shows the interaction between the system and outside entities. This context-level DFD is then "exploded" to show more detail of the system being modeled.

The Figure 3.5 shows the Level 0 Data Flow Diagram. In this level 0 DFD, two external entities have been identified i.e. log files and the recommendation. The user uploads the log file as input to the application, the log files are processed and analyzed and the recommendation for trouble shooting the application is displayed to the user.

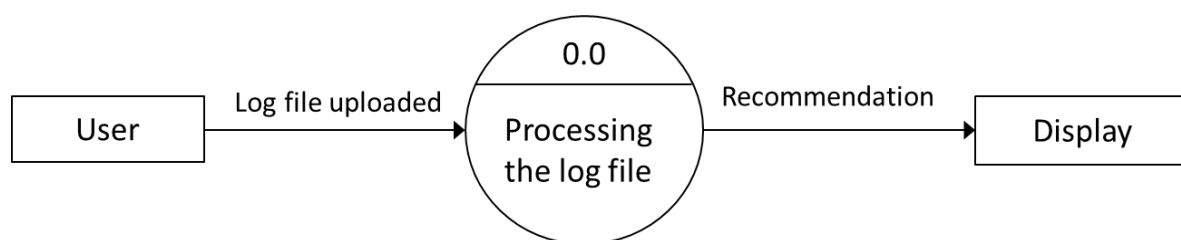
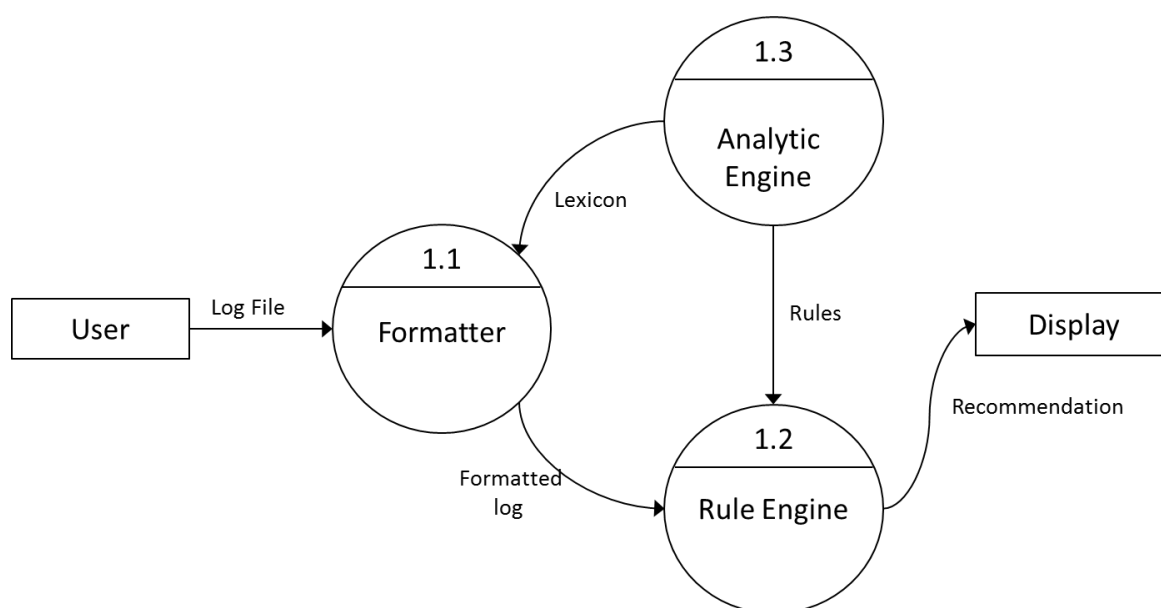


Figure 3.5 Level 0 DFD for Log Analyzer System

#### 3.6.2 Data Flow Diagram – Level 1

The Level 1 Data Flow Diagram gives more information than the level 0 Data Flow Diagram. The Figure 3.4 shows the Level 1 Data Flow Diagram. In this level 1 DFD, the process 0 shown in the level 0 DFD is broken down into detail processes. Three process for each functional module. The user input is the log file archive from the support dump and the output is the recommendation for trouble shooting the issues in the system.

The Figure 3.6 shows the Level 0 Data Flow Diagram. In this level 1 DFD, the user uploads the log files archive from the datacenter support dump. The log file is extracted and given as input to the formatter looks for the tokens in the log file from the lexicons fed by the analytic engine, thus formats the log into a universal format, this formatted log is fed as facts to the rule engine. Along with the rules from the analytic engine, the rule engine validates the formatted log and classifies the log file, depending on the classification, recommendation is provided to the user.



**Figure 3.6 Level 1 DFD for Log Analyzer System**

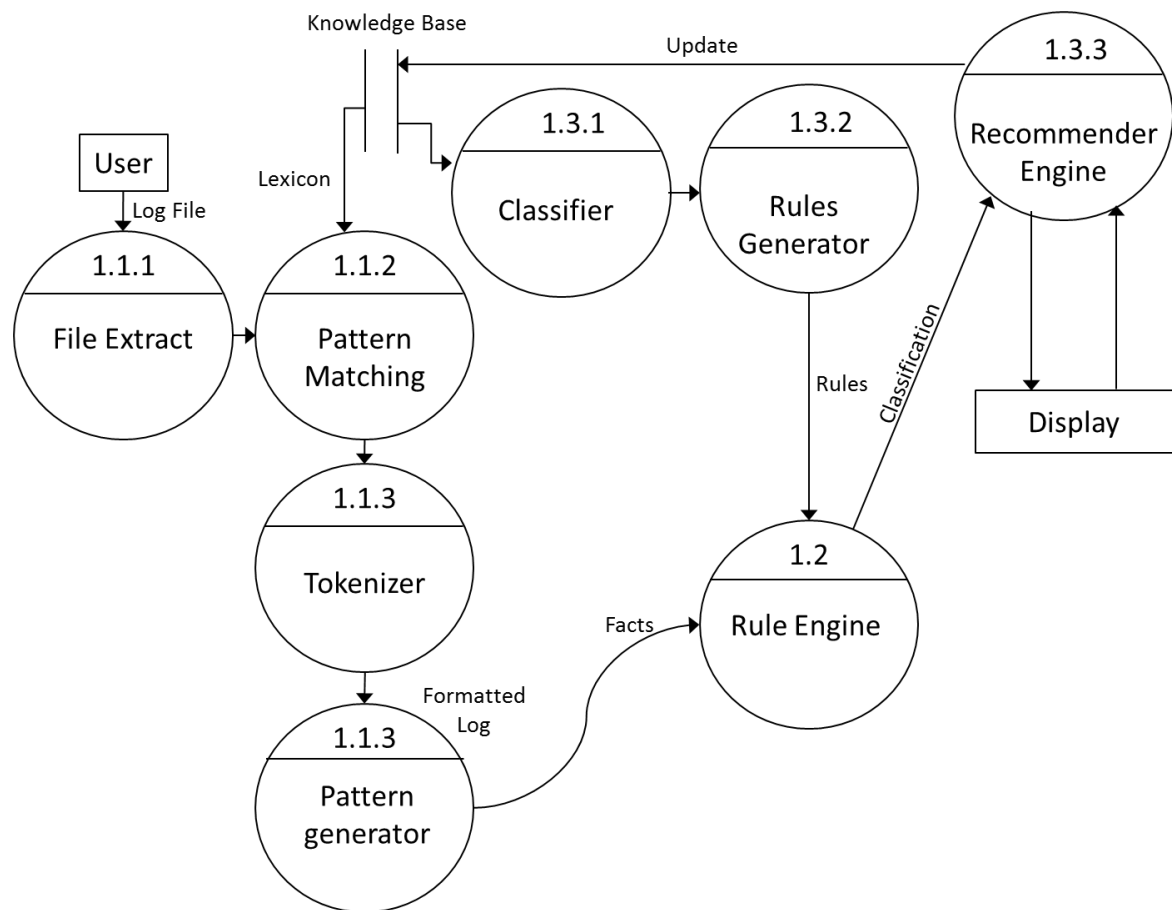
### 3.6.2 Data Flow Diagram – Level 2

The Level 2 Data Flow Diagram gives more information than the level 1 Data Flow Diagram. The Figure 3.7 shows the Level 2 Data Flow Diagram. In this level 2 the three processes are further broken into sub process.

The input log file is formatted in three steps; initially the log files are searched for the tokens that are available in the knowledgebase. The lexicon for matching the pattern is provided by the knowledgebase. The irrelevant data from the log files are filtered at this stage and the information of interest are further tokenized and thus formatted log is created. The formatted log is the object with parameters describing the issue in the log.

The formatted log is fed as facts to the rule engine. The rules to validate the log are provided by the analytics engine. The analytic engine comprises of a classifier, which

uses supervised learning decision tree algorithm to classify the log into a particular class. The tree structure generated by the classifier is represented in the form of rules which can be executed on the rule engine. Depending on the class of the issue a recommendation is provided to the user and the user provides feedback to the system which in turn updates the knowledgebase.

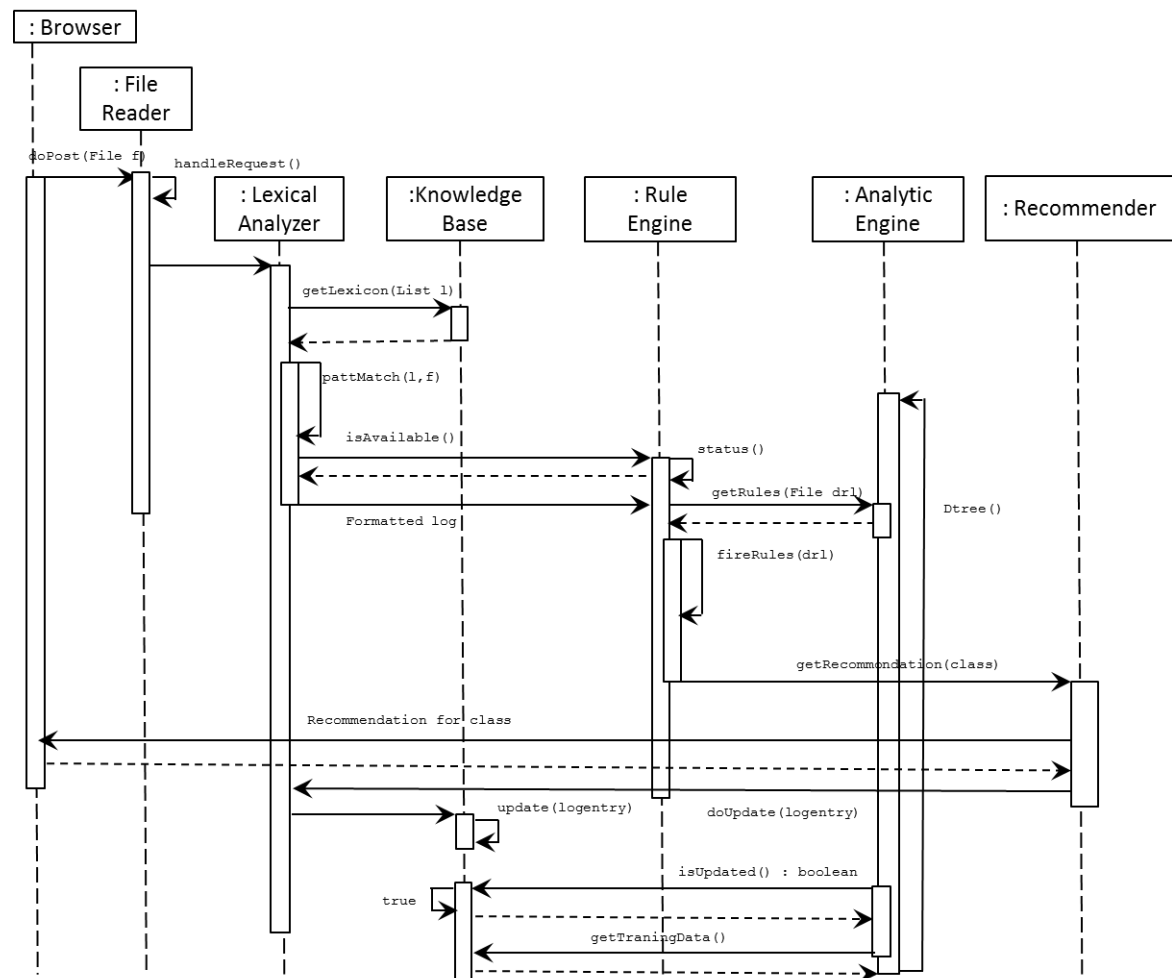


**Figure 3.7 Level 1 DFD for Log Analyzer System**

### 3.7 Sequence Diagram for Log Analyzer

A sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams, event scenarios.

Figure 3.8 shows the sequence of interaction between each module in the adaptive log analyzer.



**Figure 3.8 Sequence Diagram for Log Analyzer**

## **CHAPTER 4**

# **DETAILED DESIGN FOR KNOWLEDGE AND A RULE BASED ENGINE TO ANALYZE THE LOGS FOR TROUBLESHOOTING**

Detailed Design of a system gives in depth picture of most components described in the project. In this section, flowcharts and algorithms of each module has been described. The control flow is shown by the structure chart, the functional descriptions of which are presented in the flow chart diagrams.

This chapter presents the following:

- Structure chart for the project
- Functional Description of the modules and
- Flow Charts for the modules.

### **4.1 Structure Chart**

Structure chart shows the control flow among the modules in the system. The structure chart explains the identified modules and the interaction between the Modules. It also explains the identified Sub-Modules. Structure Chart explains the input for each modules and output generated by each module. There are three main modules in this project.

- Data Formatter module
- Rule Engine module
- Analytic Engine module

Sequence of flow in each module of Adaptive Log Analyzer system is being discussed in the following subjects of the report.

### **4.2 Functional Description of the Modules**

This section contains a detail description of software components, low-level components and other sub components of the project.

### 4.2.1 Data Formatter Module

This section contains a description of the functionality and some software component used in formatting and correlating the content in the log file [14].

- **Purpose**

The purpose is to reduce the log file size by removing the irrelevant data from the logs and extracting only the useful information from the logs.

- **Functionality**

The formatter reads the log files and correlates the data from multiple log files and along with the available training data looks for relevant patterns by pattern matching.

- **Input**

The input to this module will be the log files from the datacenter support dump. These log files are archived into zip file and uploaded. The files can be binary log files or plain text files.

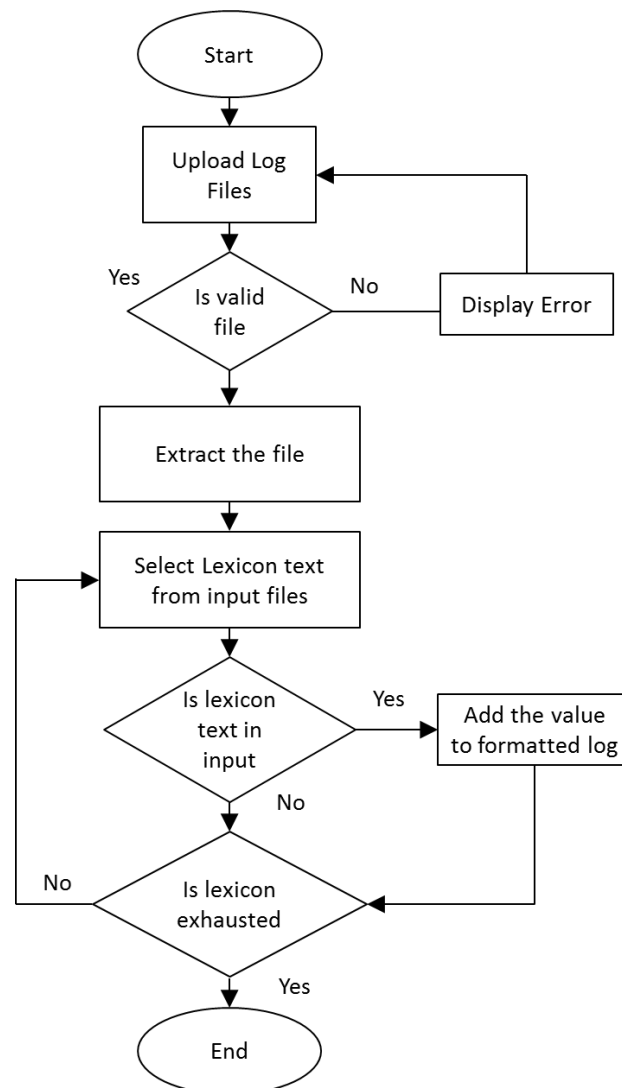
- **Output**

The output from this module would be a formatted log which is in a universal format generated by correlating the multiple log files. The information is extracted from the raw log file and an object of the formatted log is created.

- **Flow Chart**

The Formatter module consists of 3 main functions: File Uploader, File Validation, File Extraction, Lexical Analyzer and Log Formatting. The flow chart for the Formatter module is as shown in figure 4.1 the sequence of execution steps are as follows.

- a) Uploads the log file from the support dump.
- b) The file is validated to meet the specific parameter of log from support dump of datacenter.
- c) The file is extracted on to the server and made available to manually analyze the log data.
- d) The file content are read through buffer and lexically analyzed till all lexicons are exhausted.
- e) Formatted log is returned, object with all log parameters instantiated.



**Figure 4.1 Flow Chart for Data Formatter**

### 4.2.2 Rule Engine Module

This section contains a description of the functionality and some software component used in formatting and correlating the content in the log file.

- **Purpose**

The purpose is that it provides runtime environment to execute rule generated by the analytic engine. The key advantage using rules can make it easy to express solutions to difficult problems and consequently have those solutions verified. Rules are much easier to read than code.

- **Functionality**

The Rules are stored in the Production Memory and the facts that the Inference Engine matches against are kept in the Working Memory. Facts are asserted into the Working Memory where they may then be modified or retracted. A system with a large number of rules and facts may result in many rules being true for the same fact assertion; these rules are said to be in conflict. The Agenda manages the execution order of these conflicting rules using a Conflict Resolution strategy.

- **Input**

The input to this module will be the facts in the form of formatted log. The formatted log is an object instantiated by the formatter during the lexical analysis.

- **Output**

The output from this module would be a formatted log which is in a universal format generated by correlating the multiple log files. The information is extracted from the raw log file and an object of the formatted log is created.

- **Flow Chart**

Backward chaining is "goal-driven", meaning that we start with a conclusion which the engine tries to satisfy. The rule engine executes the rules. Flow chart for the Rule Engine is as shown in figure 4.2 the sequence of execution steps are as follows.

- a) Agenda maintains set of rules that are able to execute, its job is to schedule that execution in a deterministic order.
- b) Examine the working memory and goals are known true in knowledge base.
- c) Determine the next possible rules to fire by checking conclusions and goals.
- d) Rule Matches and Conflict Sets.
- e) Inference the result set.
- f) Backward chaining is "goal-driven", meaning that we start with a conclusion which the engine tries to satisfy.
- g) If it can't it then searches for conclusions that it can satisfy; these are known as sub goals, which will help satisfy some unknown part of the current goal.
- h) It continues this process until either the initial conclusion is proven or there are no more sub goals.
- i) For each rule condition, recursively back chain with condition as goal
- j) If no rule is found in the working set then exit.



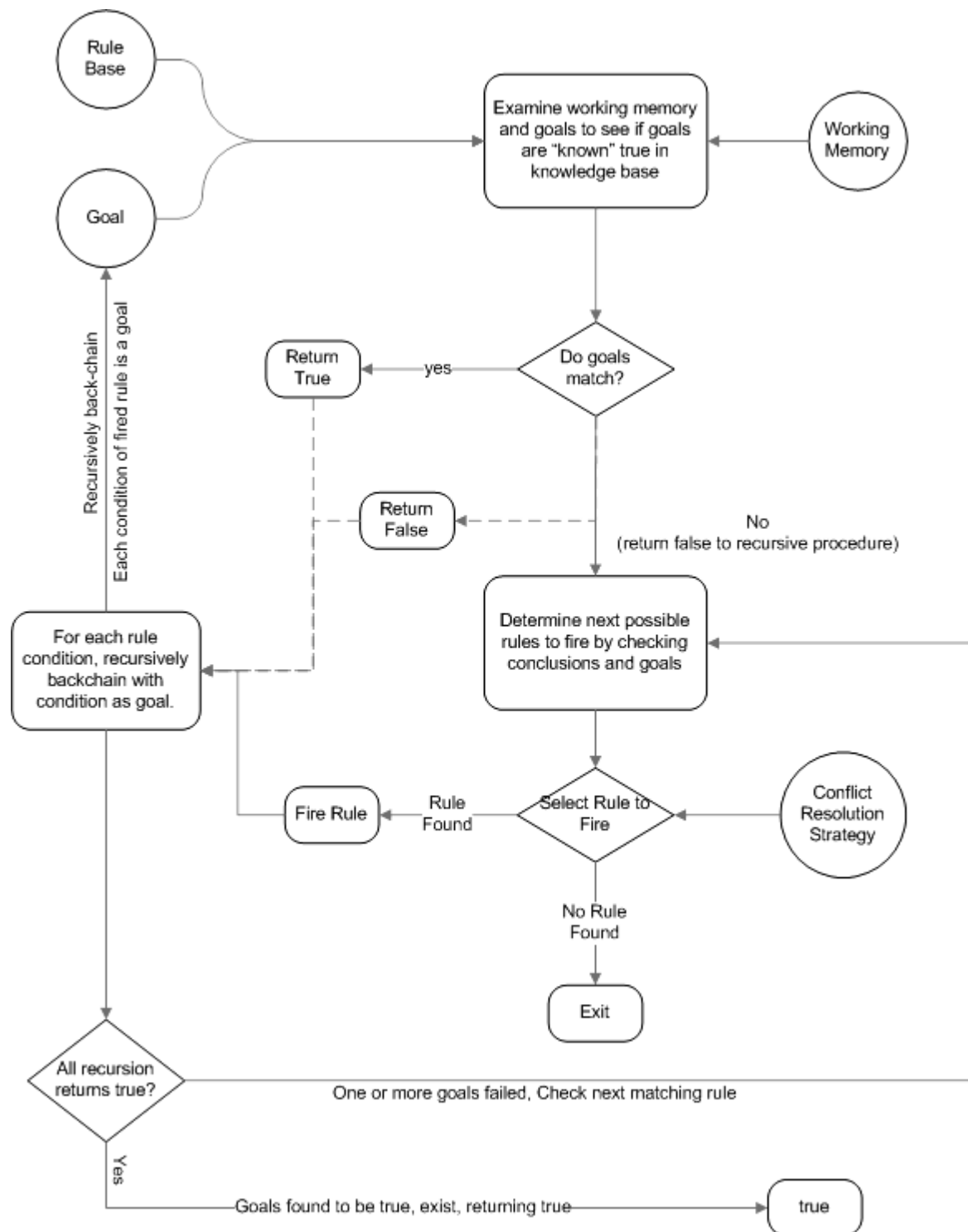


Figure 4.2 Flow Chart for Rule Engine Backward Chaining

### 4.2.3 Analytic Engine Module

This section contains a description of the functionality and some software component used in constructing the decision tree and deriving the rule from the labeled training data and providing recommendation depending on the classification [15].

- **Purpose**

The purpose is to generate rules from the training data and in keeping the knowledgebase updated by the user feedback.

- **Functionality**

Constructs the decision tree from the, available training data. And derive rules to validate the incoming log files.

- **Input**

The input to this module will be the labeled training data in the knowledge base, this knowledge base is updated every time the user sends the feedback to the system and new rules are derived accordingly.

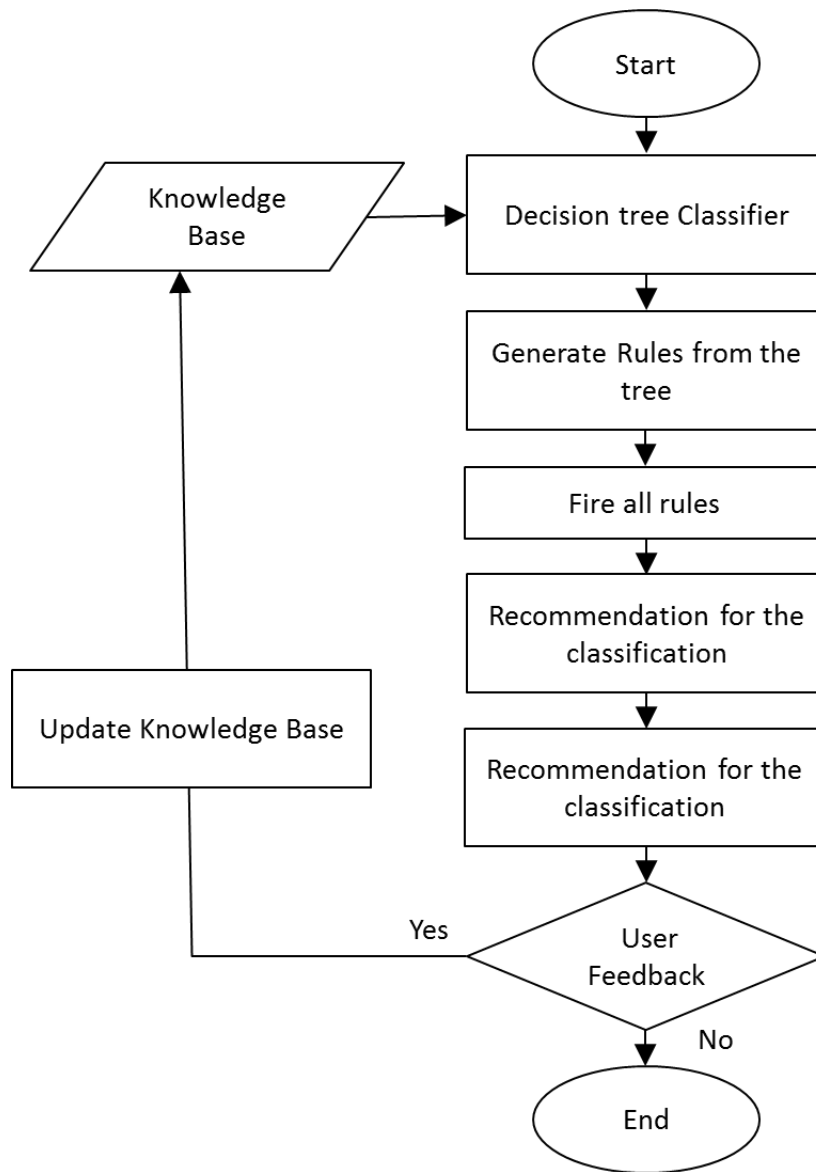
- **Output**

The output from this module would be a rule file with \*.drl extension. This file is executed in the rule engine. And the classification provided by the rule engine is used to provide recommendation to the user.

- **Flow Chart**

The Formatter module consists of 4 main functions: Classification of the issue, Generating the Rules, Recommendation by classification and updating the Knowledgebase. The flow chart for the Formatter module is as shown in figure 4.1 the sequence of execution steps are as follows.

- a) Data is classified using decision tree learning
- b) The rules are derived from the decision tree.
- c) Recommendation for the classification
- d) Updating the knowledge base depending on the user input.

**Figure 4.3 Flow Chart for Analytics Engine**

## **CHAPTER 5**

# **IMPLEMENTATION OF KNOWLEDGE AND A RULE BASED ENGINE TO ANALYZE THE LOGS FOR TROUBLESHOOTING**

Implementation of any software is always preceded by important decisions regarding selection of the platform, the language used, etc. These decisions are often influenced by several factors such as the real environment in which the system works, the speed that is required, the security concerns, other implementation specific details etc.,

Java Eclipse Integrated Development Environment (IDE) is used for the development of the application. Java is used as a programming language. The strengths of the java as a programming tool for the coders and its facilities offered to users to minimize the logic flow by using standard libraries have been discussed in this chapter.

### **5.1 Programming Language Selection**

Java is a programming language used in the project, originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform [15]. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to byte code (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture. Java is a general-purpose, concurrent, class-based, object-oriented language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere". Java is currently one of the most popular programming languages in use, and is widely used from application software to web applications. "Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, multithreaded, and dynamic." [10].

Java consists of following features:

- **Productivity:** The top reason Java has become popular is because of the increased productivity of Java programmers. It is claimed, and my experience is in agreement that Java programmers have about double the productivity of C/C++ programmers.
- **GUI:** Java a good, portable library for a Graphical User Interface (GUI). Most programming languages supply only a text mode interface.
- **Internet:** Java lets you easily use the Internet. Most languages were designed before the Internet was born!
- **Portability:** Java programs can run on many different machines (Intel, Sparc, PowerPC,) and many different operating systems (Windows, Unix, Macintosh,). You can move a C program if it is written very carefully, but it is usually difficult or impossible. In contrast, it easy to move most Java programs.
- **Reliability:** Java programs are more reliable because Java doesn't have very dangerous things like C/C++'s pointer arithmetic. Java also checks array bounds and other error-prone operations. Memory management is much safer because Java does automatic garbage collection.
- **Libraries:** Java has a very large number of packages which extend the language. Therefore it is unnecessary to call the operating system directly.
- **OOP:** Object-oriented programming features (inheritance, encapsulation, and polymorphism) make many programs, especially large programs, easier to write.
- **Large Programs:** Java supports large programming projects with object-oriented programming, packages, and components.

### 5.1.1 Java Server Pages

**Java Server Pages (JSP)** is a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types. Released in 1999 by Sun Microsystems, JSP is similar to PHP, but it uses the Java programming language.

To deploy and run Java Server Pages, a compatible web server with a servlet container, such as Apache Tomcat or Jetty, is required. Architecturally, JSP may be viewed as a

---

high-level abstraction of Java servlets. JSPs are translated into servlets at runtime; each JSP's servlet is cached and re-used until the original JSP is modified.

JSP can be used independently or as the view component of a server-side model–view–controller design, normally with JavaBeans as the model and Java servlets (or a framework such as Apache Struts) as the controller. This is a type of Model 2 architecture.

JSP allows Java code and certain pre-defined actions to be interleaved with static web markup content, with the resulting page being compiled and executed on the server to deliver a document. The compiled pages, as well as any dependent Java libraries, use Java byte code rather than a native software format. Like any other Java program, they must be executed within a Java virtual machine (JVM) that integrates with the server's host operating system to provide an abstract platform-neutral environment.

JSPs are usually used to deliver HTML and XML documents, but through the use of Output Stream, they can deliver other types of data as well.

The Web container creates JSP implicit objects like `pageContext`, `servletContext`, `session`, `request` & `response`.

JSP pages use several delimiters for scripting functions. The most basic is `<% ... %>`, which encloses a JSP *scriptlet*. A scriptlet is a fragment of Java code that is run when the user requests the page. Other common delimiters include `<%= ... %>` for *expressions*, where the value of the expression is placed into the page delivered to the user, and *directives*, denoted with `<%@ ... %>`

Java code is not required to be complete or self-contained within its scriptlet element block, but can straddle markup content providing the page as a whole is syntactically correct. For example, any Java `if/for/while` blocks opened in one scriptlet element must be correctly closed in a later element for the page to successfully compile. Markup which falls inside a split block of code is subject to that code, so markup inside an *if* block will only appear in the output when the *if* condition evaluates to true; likewise, markup inside a loop construct may appear multiple times in the output depending upon how many times the loop body runs.

---

The following would be a valid for loop in a JSP page:

```
<p>Counting to three:</p>

<% for (inti=1; i<4; i++) { %>

<p>This number is <%= i %>.</p>

<% } %>

<p>Done counting.</p>
```

### 5.1.2 DRools

Business Rule Management Systems build additional value on top of a general purpose Rule Engine by providing business user focused systems for rule creation, management, deployment, collaboration, and analysis and end user tools. Further adding to this value is the fast evolving and popular methodology "Business Rules Approach", which is helping to formalize the role of Rule Engines in the enterprise.

The term Rule Engine is quite ambiguous in that it can be any system that uses rules, in any form that can be applied to data to produce outcomes. This includes simple systems like form validation and dynamic expression engines. The book "How to Build a Business Rules Engine (2004)" by Malcolm Chisholm exemplifies this ambiguity. The book is actually about how to build and alter a database schema to hold validation rules. The book then shows how to generate VB code from those validation rules to validate data entry. This, while a very valid and useful topic for some, caused quite a surprise to this author, unaware at the time in the subtleties of Rules Engines' differences, who was hoping to find some hidden secrets to help improve the Drools engine. JBoss jBPM uses expressions and delegates in its Decision nodes which control the transitions in a Workflow. At each node it evaluates there is a rule set that dictates the transition to undertake, and so this is also a Rule Engine. While a Production Rule System is a kind of Rule Engine and also an Expert System, the validation and expression evaluation Rule Engines mentioned previously are not Expert Systems.

A Production Rule System is Turing complete, with a focus on knowledge representation to express propositional and first order logic in a concise, non-ambiguous and declarative

manner. The brain of a Production Rules System is an Inference Engine that is able to scale to a large number of rules and facts. The Inference Engine matches facts and data against Production Rules - also called Productions or just Rules - to infer conclusions which result in actions. A Production Rule is a two-part structure using First Order Logic for reasoning over knowledge representation.

```
when  
  
<conditions>  
  
then  
  
<actions>;
```

Drools implements and extends the Rete algorithm; Leaps used to be provided but was retired as it became unmaintained. The Drools Rete implementation is called ReteOO, signifying that Drools has an enhanced and optimized implementation of the Rete algorithm for object oriented systems. Other Rete based engines also have marketing terms for their proprietary enhancements to Rete, like RetePlus and Rete III. The most common enhancements are covered in "Production Matching for Large Learning Systems (Rete/UL)" (1995) by Robert B. Doorenbos.

The Rules are stored in the Production Memory and the facts that the Inference Engine matches against are kept in the Working Memory. Facts are asserted into the Working Memory where they may then be modified or retracted. A system with a large number of rules and facts may result in many rules being true for the same fact assertion; these rules are said to be in conflict. The Agenda manages the execution order of these conflicting rules using a Conflict Resolution strategy.

## **5.2 Platform Selection**

Windows platform is chosen to implement the modules faster using java technologies and speed up to the markets. The reason for choosing windows operating system is mainly due to user friendliness and familiarity in use and Java version compatible to windows is used for developing the application.

## **5.3 Code Conventions**

Code conventions make programs more understandable by making them easier to read and to manage the code. They can also give information about the function of the



identifier—for example, whether it's a constant, package, or class—which can be helpful in understanding the code. For the conventions to work, everyone writing software must conform to the code conventions.

### **5.3.1 Naming conventions**

Naming conventions make programs more understandable by making them easier to read. They can also give information about the function of the identifier—for example, whether it's a constant, package, or class—which can be helpful in understanding the code. For the conventions to work, everyone writing software must conform to the code conventions. The naming rules for the identifiers are explained below:

- **Package:** The prefix of a unique package name is always written in all lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981. Subsequent components of the package name vary according to an organization's own internal naming conventions [16]. Such conventions might specify that certain directory name components be division, department, project, machine, or login names. Examples: com.sun.eng, com.apple.quicktime.v2.
- **Classes:** Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML). Example: class Enterprise Window, class Login Form.
- **Interfaces:** Interface names should be capitalized like class names. Example: ActionListener, KeyListener.
- **Methods:** Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized. Example: run(), getBackground().
- **Variables:** Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore \_ or dollar sign \$ characters, even though both are allowed. Variable names should be short yet meaningful. The

choice of a variable name should be mnemonic— that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary “throwaway” variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.

Example: `inti`, `char c`, `float myWidth`.

- Constants: The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores (“\_”). (ANSI constants should be avoided, for ease of debugging). Example: `static final int MIN_WIDTH = 4`.

### 5.3.2 File Organization

A file consists of sections that should be separated by blank lines and an optional comment identifying each section. Files longer than 2000 lines are cumbersome and should be avoided.

#### 5.3.2.1 Java Source Files

Each Java source file contains a single public class or interface. When private classes and interfaces are associated with a public class, you can put them in the same source file as the public class. The public class should be the first class or interface in the file.

Java source files have the following ordering:

- Beginning comments
- Package and Import statements
- Class and interface declarations

#### 5.3.2.2 Beginning Comments

All source files should begin with a c-style comment that lists the class name, version information, date, and copyright notice:

```
/*  
  
 * Class name  
  
 *  
  
 * Version information
```

---

```
*  
  
* Date  
  
*  
  
* Copyright notice  
  
*/
```

### 5.3.2.3 Package and Import Statements

The first non-comment line of most Java source files is a package statement. After that, import statements can follow. For example:

```
package java.awt;  
  
import java.awt.event;
```

### 5.3.2.4 Class and Interface Declarations

The following describes the parts of a class or interface declaration, in the order that they should appear.

- **Class/interface documentation comment (/\*\*...\*/):** Documentation comments describe Java classes, interfaces, constructors, methods, and fields. Each doc comment is set inside the comment delimiters `/**...*/`, with one comment per class, interface, or member. This comment should appear just before the declaration:

```
/**  
 * The Example class provides ...  
 */  
  
public class Example { ...  
  
    class or interface statement
```

- **Class/interface implementation comment (/\*...\*/), if necessary:** This comment should contain any class-wide or interface-wide information that wasn't appropriate for the class/interface documentation comment.

- **Class (static) variables:** First the public class variables, then the protected, then package level (no access modifier), and then the private.
- **Instance variables:** First public, then protected, then package level (no access modifier), and then private.
- **Constructors:** Constructor in a class is a special type of subroutine called at the creation of an object. It is called a constructor because it constructs the values of data members of the class.
- **Methods:** These methods should be grouped by functionality rather than by scope or accessibility. For example, a private class method can be in between two public instance methods. The goal is to make reading and understanding the code easier.

### 5.3.2.5 Class declaration

This section discusses about class declaration with an example

```
class MyClass {  
    //field, constructor, and method declarations  
}
```

This is a class declaration. The class body (the area between the braces) contains all the code that provides for the life cycle of the objects created from the class: constructors for initializing new objects, declarations for the fields that provide the state of the class and its objects, and methods to implement the behavior of the class and its objects.

The preceding class declaration is a minimal one—it contains only those components of a class declaration that are required. You can provide more information about the class, such as the name of its superclass, whether it implements any interfaces, and so on, at the start of the class declaration. For example,

```
class MyClass      extends      MySuperClass      implements  
YourInterface {  
    //field, constructor, and method declarations}
```

Means that MyClass is a subclass of MySuperClass and that it implements the YourInterface interface.

You can also add modifiers like public or private at the very beginning—so you can see that the opening line of a class declaration can become quite complicated. The modifiers public and private, which determine what other classes can access MyClass,

In general, class declarations can include these components, in order:

- Modifiers such as public, private, and protected.
- The class name, with the initial letter capitalized by convention.
- The class body, surrounded by braces, { }.
- The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
- A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.

### 5.3.2 Comments

To comprehend any programming language, there are several kind of comments which are used. These comments are advantageous in the sense that they make the programmer feel convenient to grasp the logic of the program. Although these comments are ignored by the Java compiler, they are included in the program for the convenience of the user to understand it. To provide the additional information about the code, use comments. These comments give the overview of the code in the form of the information which is not available in the code itself.

There are three types of comments used in Java. These are:

#### a) // text

To add a comment to the program, we can use two slashes characters i.e. //. The line starting from slashes to the end is considered as a comment. We can write only a single line comment use these slashes. For instance

```
// This comment extends to the end of the line.
```

```
// This type of comment is called a "slash-slash"  
comment
```

#### b) /\* text \*/

To add a comment of more than one line, we can precede our comment using `/*`. The precise way to use this is to start with delimiter `/*` and end with delimiter `*/`. Everything in between these two delimiters is discarded by the Java compiler. For instance

```
/* This comment, a "slash-star" comment, includes
multiple lines.

* It begins with the slash-star sequence (with no
space between

* the '/' and '*' characters) and extends to the
star-slash sequence.

*/
```

Slash-star comments may also be placed between any Java tokens:

```
inti = /* maximum integer */ Integer.MAX_VALUE;
```

**c) `/** documentation */`**

This is a special type of comment that indicates documentation comment. This type of comment is readable to both, computer and human. To start the comment, use `/**` instead of `/*` and end with `*/`. This type of comment is a documentation which is interpreted as an official document on how the class and its public method work. For instance

```
/**

* These are used to extract documentation from the
Java source.

*/
```

## 5.4 Programming Techniques

This section discusses about the packages used:

### 5.4.1 Packages used

This section discusses about the packages that is used for developing the application that is provided by java.

The packages used are

- **java.io:** Provides for system input and output through data streams, serialization and the file system.
- **java.util:** Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).
- **org.jfree.chart:** Provides the classes for creating professional quality charts in their applications.
- **java.sql:** Provides the API for accessing and processing data stored in a data source (usually a relational database) using the Java programming language.
- **org.drools:** Drools & jBPM provide a knowledge-centric API, where rules and processes are program reducing the Java programming language.
- **javax.servlet:** Provides the classes and interfaces that describe and define the contracts between a servlet class and the runtime environment provided for an instance of such a class by a conforming servlet container.

The **java.io** Provides for system input and output through data streams, serialization and the file system. Unless otherwise noted, passing a null argument to a constructor or method in any class or interface in this package will cause a `NullPointerException` to be thrown.

The **java.util** Java Utility package is one of the most commonly used packages in the java program. The Utility Package of Java consist of the following components collections framework, legacy collection classes, event model, date and time facilities, internationalization, miscellaneous utility classes such as string tokenizer, random-number generator and bit array

The **org.jfree.chart** package can be used to create charts to display the results pertaining to the project data. The data can be rendered upon iterations and the results can be displayed as charts.

The **java.sql** Provides the API for accessing and processing data stored in a data source (usually a relational database) using the Java programming language. This API includes a framework whereby different drivers can be installed dynamically to access different data sources. Although the JDBC API is mainly geared to passing SQL statements to a

database, it provides for reading and writing data from any data source with a tabular format. The reader/writer facility, available through the `javax.sql.RowSet` group of interfaces, can be customized to use and update data from a spread sheet, flat file, or any other tabular data source.

The **org.droolsfactory** classes, with static methods, provide instances of the above interfaces. A pluggable provider approach is used to allow provider implementations to be wired up to the factories at runtime. 'kbuilder', 'kbase', 'ksession' are the variable identifiers often used, the k prefix is for 'knowledge'.

The **javax.servlet** package contains a number of classes and interfaces that describe and define the contracts between a servlet class and the runtime environment provided for an instance of such a class by a conforming servlet container.

## 5.4 Java Eclipse IDE

Eclipse began as an IBM Canada project. Object Technology International (OTI), which had previously marketed the Smalltalk-based VisualAge family of integrated development environment (IDE) products, developed the new product as a Java-based replacement.[4] In November 2001, a consortium was formed with a board of stewards to further the development of Eclipse as open-source software. It is estimated that IBM had already invested close to \$40 million by that time. The original members were Borland, IBM, Merant, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft and WebGain. The number of stewards increased to over 80 by the end of 2003. In January 2004, the Eclipse Foundation was created[17].

Eclipse uses plug-ins to provide all the functionality within and on top of the runtime system. Its runtime system is based on Equinox, an implementation of the OSGi core framework specification. In addition to allowing the Eclipse Platform to be extended using other programming languages such as C and Python, the plug-in framework allows the Eclipse Platform to work with typesetting languages like LaTeX,[28] networking applications such as telnet and database management systems. The plug-in architecture supports writing any desired extension to the environment, such as for configuration management. Java and CVS support is provided in the Eclipse SDK, with support for other version control systems provided by third-party plug-ins.



With the exception of a small run-time kernel, everything in Eclipse is a plug-in. This means that every plug-in developed integrates with Eclipse in exactly the same way as other plug-ins; in this respect, all features are "created equal".[citation needed] Eclipse provides plug-ins for a wide variety of features, some of which are through third parties using both free and commercial models. Examples of plug-ins include for UML, for Sequence and other UML diagrams, a plug-in for DB Explorer, and many others.

The Eclipse SDK includes the Eclipse Java development tools (JDT), offering an IDE with a built-in incremental Java compiler and a full model of the Java source files. This allows for advanced refactoring techniques and code analysis. The IDE also makes use of a workspace, in this case a set of metadata over a flat filespace allowing external file modifications as long as the corresponding workspace "resource" is refreshed afterwards.

Eclipse implements widgets through a Java toolkit called SWT, whereas most Java applications use the Java standard Abstract Window Toolkit (AWT) or Swing. Eclipse's user interface also uses an intermediate graphical user interface layer called JFace, which simplifies the construction of applications based on SWT.

The platform offers reusable services common to desktop applications, allowing developers to focus on the logic specific to their application. Among the features of the platform are:

- User interface management (e.g. menus and toolbars)
- User settings management
- Storage management (saving and loading any kind of data)
- Window management
- Wizard framework (supports step-by-step dialogs)
- Integrated development tools

## **5.5 Difficulties Encountered and Strategies Used to Tackle**

The implementation difficulties encountered during the project development and strategies used to solve these difficulties have been explained in this section.

### **5.5.1 Traversing the tree to generate rules**

The decision tree constructed from the training data had to be traversed in order to generate rules required to be executed on rule engine. The depth first search was used to

traverse the tree, but the results were unsymmetrical rules that could not be executed on rule engine. The problem was solved by traversing the from the root node till the leaf node for every classification in the leaf node.

### **5.5.2 Integrating drools rule engine**

The drools which provide runtime for executing the rules couldn't be deployed on the tomcat 7.0.53 due to API conflicts between org.apache.catalina.core and the drools API. Hence the drools had to be deployed as a RESTful services with the implementation of Camel server.

## CHAPTER 6

# SOFTWARE TESTING FOR KNOWLEDGE AND A RULE BASED ENGINE TO ANALYZE THE LOGS FOR TROUBLESHOOTING

At testing stage of this project, defects/errors were discovered by testing individual program components. Testing was focused on establishing functional requirements and on system behavior in given scenario. The test cases are selected to ensure that the system behavior can be examined in all possible combinations of conditions.

Accordingly, expected behavior of the system under different combinations is given. Therefore test cases are selected which have inputs and the outputs are on expected lines, inputs that are not valid and for which suitable messages must be given and inputs that do not occur very frequently which can be regarded as special cases.

- **Verification:** Verification is the process to make sure the product satisfies the conditions imposed at the start of the development phase. In other words, to make sure the product behaves the way it wants it to.
- **Validation:** Validation is the process to make sure the product satisfies the specified requirements at the end of the development phase. In other words, to make sure the product is built as per customer requirements.

In this chapter, several test cases have been explained with the underlying log analyzer system.

### 6.1 Test Strategy

Test strategy is an outline that describes the testing approach for the product or system under development. It also describes which types of test are to be performed, and which entry and exit criteria apply. The test strategy also describes the test level to be performed. There are primarily three levels of testing: unit testing, integration testing, and system testing. In most software development organizations, the developers perform unit testing. Individual testers or test teams perform integration and system testing.

Environment requirements play a very important role, while describing the test strategy. It describes what operating systems are used and automated testing. Depending on the nature of the testing, either manual or automated or a combination of both is used. For the project, manual approach is used. Thus the test strategy document is a high level document which defines the testing approach to achieve testing objectives.

## **6.2 Levels of Testing**

Different levels of testing are used in the software testing activity. Tests are grouped by where they are added in the software development process, or by the level of specificity of the test. The main levels which are tested here are unit testing, integration testing, system testing and acceptance testing, which are briefly described below:

### **6.2.1 Unit Testing**

The basic building blocks of unit testing are test cases — single scenarios that must be set up and checked for correctness.

- Input values: Write test cases for each of the identified inputs (positive & negative) accepted by the Unit.
- Expected Functionality: Every functionality that is expected to be in the unit is covered.
- Output values: Write test cases which will produce all types of output values that are expected from the module / unit.
- Path coverage: If the unit have conditional processing that results in various paths, then test cases needs to be written to cover each of these paths.
- Abnormal terminations: Behavior of the module or unit in case of abnormal termination should be tested.
- Error messages: Error messages / warnings needs to be checked. These should be short, precise and self-explanatory. They should be properly phrased and free of grammatical mistakes.

Each component of the Image acquisition, Image pre-processing and Image feature extraction module is tested independently and the test cases are tabulated in the following tables from Table 6.1 – Table 6.9.

**6.2.1.1 File uploader**

From the Table 6.1 represents the unit case for uploading the file into the system.

**Table 6.1: Unit Test Case 1 for File Uploading**

|                        |                             |
|------------------------|-----------------------------|
| <b>Test Case ID</b>    | Unit Test case 1            |
| <b>Description</b>     | Verify file uploading       |
| <b>Input</b>           | File uploaded in zip format |
| <b>Expected Output</b> | File successfully uploaded  |
| <b>Actual Output</b>   | File successfully uploaded  |
| <b>Remarks</b>         | Passed                      |

From the Table 6.1 represents the unit case for uploading the file into the system. The input for the system is file in zip format from the datacenter support dump. Actual output is same as expected output. Therefore, the result of the unit test case 1 passed.

**6.2.1.2 File Extractor**

From the Table 6.2 represents the unit case for extracting the uploaded file into the dashboard of the system. The input for the system is file in zip format from the datacenter support dump.

**Table 6.2: Unit Test Case 2 for File Extraction**

|                        |                             |
|------------------------|-----------------------------|
| <b>Test Case ID</b>    | Unit Test case 2            |
| <b>Description</b>     | Verify file Extraction      |
| <b>Input</b>           | File uploaded in zip format |
| <b>Expected Output</b> | File successfully Extracted |
| <b>Actual Output</b>   | File successfully Extracted |
| <b>Remarks</b>         | Passed                      |

From the Table 6.2 represents the unit case for extracting the uploaded file into the dashboard of the system. The input for the system is file in zip format from the datacenter support dump. Actual output is same as expected output. Therefore, the result of the unit test case 2 passed.

### 6.2.1.3 File Browser and File Reader

From the Table 6.3 represents the unit case for visualizing file into the dashboard of the system. The input for the system is file path where log files are extracted.

**Table 6.3: Unit Test Case 3 for File Browser and Reader**

|                        |                                       |
|------------------------|---------------------------------------|
| <b>Test Case ID</b>    | Unit Test case 3                      |
| <b>Description</b>     | Verify file browser and reader        |
| <b>Input</b>           | Extracted file path                   |
| <b>Expected Output</b> | Visualization of the file tree in GUI |
| <b>Actual Output</b>   | Visualization of the file tree in GUI |
| <b>Remarks</b>         | Passed                                |

From the Table 6.3 represents the unit case for visualizing file into the dashboard of the system. The input for the system is file path where log files are extracted. Actual output is same as expected output. Therefore, the result of the unit test case 3 passed.

### 6.2.1.4 Lexical Analyzer

From the Table 6.4 represents the unit case for structuring the log files into a universal format. The input for the system is file path where log files are extracted.

**Table 6.4: Unit Test Case 4 for Lexical Analyzer**

|                        |                                    |
|------------------------|------------------------------------|
| <b>Test Case ID</b>    | Unit Test case 4                   |
| <b>Description</b>     | Verify file Lexical Analyzer       |
| <b>Input</b>           | Unstructured log files             |
| <b>Expected Output</b> | Structured log in universal format |
| <b>Actual Output</b>   | Structured log in universal format |
| <b>Remarks</b>         | Passed                             |

From the Table 6.4 represents the unit case for structuring the log files into a universal format. The input for the system is file path where log files are extracted. Actual output is same as expected output. Therefore, the result of the unit test case 4 passed.

#### **6.2.1.5 Rule Engine**

From the Table 6.5 represents the unit case for rule engine. The input for the system is the rules in drl file and facts derived from the formatted log.

**Table 6.5: Unit Test Case 5 for Rule Engine**

|                        |                    |
|------------------------|--------------------|
| <b>Test Case ID</b>    | Unit Test case 5   |
| <b>Description</b>     | Verify Rule Engine |
| <b>Input</b>           | Rules and facts    |
| <b>Expected Output</b> | Inferences         |
| <b>Actual Output</b>   | Inferences         |
| <b>Remarks</b>         | Passed             |

From the Table 6.5 represents the unit case for rule engine. The input for the system is the rules in drl file and facts derived from the formatted log. Actual output is same as expected output. Therefore, the result of the unit test case 5 passed.

### 6.2.1.6 Analytic Engine

From the Table 6.6 represents the unit case for rule engine. The input for the system is the training data in the form of tables collected during the training phase and the output is the decision tree in the form of JBoss Rules.

**Table 6.6: Unit Test Case 5 for Analytic Engine**

|                        |                          |
|------------------------|--------------------------|
| <b>Test Case ID</b>    | Unit Test case 6         |
| <b>Description</b>     | Verify Analytic Engine   |
| <b>Input</b>           | Training data            |
| <b>Expected Output</b> | Rules from decision tree |
| <b>Actual Output</b>   | Rules from decision tree |
| <b>Remarks</b>         | Passed                   |

From the Table 6.6 represents the unit case for rule engine. The input for the system is the training data in the form of tables collected during the training phase and the output is the decision tree in the form of JBoss Rules. Actual output is same as expected output. Therefore, the result of the unit test case 5 passed.

### 6.2.2 Integration Testing

The second level of testing is called integration testing. In this, many unit test modules are combined into subsystems, which are then tested. Once individual program components have been tested, they must be integrated to create a partial or complete system. This integration process involves building the system and testing the resultant system for problems that arise from component interactions.

Integration tests should be developed from the system specification and integration testing should begin as soon as usable versions of some of the system components are available. The main difficulty that arises in integration testing is localizing errors that are discovered during the process.



The strategy indicates the features to be tested when the integrated modules function together. The functionality of the modules taken together is tested. The purpose of testing is to check whether the integrated modules perform as expected. The pass or fail criteria is the matching of the expected and the actual outputs of the integrated modules.

#### 6.2.2.1 File Processing

From the Table 7.13 represents the unit case for file uploading extracting and browsing, the input is the file from the support dump of the datacenter and the output is the files displayed in the dashboard of the GUI.

**Table 6.7: Integration Test Case 7 for File uploading extracting and browsing**

|                        |   |
|------------------------|---|
| <b>Test Case ID</b>    | Integration Test Case 7                                       |
| <b>Description</b>     | Integration testing of File uploading extracting and browsing |
| <b>Input</b>           | File in ZIP format from Support dump                          |
| <b>Expected Output</b> | Files are displayed on dashboard of GUI                       |
| <b>Actual Output</b>   | Files are displayed on dashboard of GUI                       |
| <b>Remarks</b>         | Passed  |

From the Table 7.13 represents the unit case for file uploading extracting and browsing, the input is the file from the support dump of the datacenter and the output is the files displayed in the dashboard of the GUI. Actual output is same as expected output. Therefore, the result of the integration test case 7 passed.

#### 6.2.2.2 Decision System

From the Table 7.13 represents the unit case for file uploading extracting and browsing, the input is the file from the support dump of the datacenter and the output is the files displayed in the dashboard of the GUI.

**Table 6.8: Integration Test Case 7 for formatter, rule engine and analytic engine**

|                        |   |
|------------------------|---|
| <b>Test Case ID</b>    | Integration Test Case 8   |
| <b>Description</b>     | Integration testing of formatter, rule engine and analytic engine |
| <b>Input</b>           | Unstructured log  |
| <b>Expected Output</b> | Rules written into DRL file                                       |
| <b>Actual Output</b>   | Rules written into DRL file                                       |
| <b>Remarks</b>         | Passed  |

From the Table 7.13 represents the unit case for file uploading extracting and browsing, the input is the file from the support dump of the datacenter and the output is the files displayed in the dashboard of the GUI. Actual output is same as expected output. Therefore, the result of the unit test case 1 passed.

### 6.2.2.3 Recommender System

From the Table 7.13 represents the unit case for file uploading extracting and browsing, the input is the file from the support dump of the datacenter and the output is the files displayed in the dashboard of the GUI.

**Table 6.9: Integration Test Case 7 for rule engine, analytic engine and recommender**

|                        |   |
|------------------------|---|
| <b>Test Case ID</b>    | Integration Test Case 7   |
| <b>Description</b>     | Integration testing of rule engine, analytic engine and recommender |
| <b>Input</b>           | Structured log in universal format                                  |
| <b>Expected Output</b> | Display the recommendation  |
| <b>Actual Output</b>   | Display the recommendation  |
| <b>Remarks</b>         | Passed  |

From the Table 7.13 represents the unit case for file uploading extracting and browsing, the input is the file from the support dump of the datacenter and the output is the files displayed in the dashboard of the GUI. Actual output is same as expected output. Therefore, the result of the unit test case 1 passed.

## CHAPTER 7

# RESULTS OF KNOWLEDGE AND A RULE BASED ENGINE TO ANALYZE THE LOGS FOR TROUBLESHOOTING

Evaluation metrics are used to evaluate the effectiveness of application developed and to justify theoretical and developments of these systems. This chapter explains the results of the experiments carried out and the analysis made using the results. It explains in detail about the evaluation metrics used followed by performance analysis and interference from the result.

### 7.1 Evaluation Metrics

In the process of evaluation and analysis of data analysis becomes more important. During the measurement, classification is performed on each of these feature set which describes the performance of each feature sets and each feature sets are compared to get which feature set provides more classification accuracy. The metrics used for network performance evaluation are:

- **Kappa statistic:** Kappa statistic is a term for several similar measures of agreement used with categorical data . Typically it is used in assessing the degree to which two or more raters, examining the same data, agree when it comes to assigning the data to categories
- **Mean Absolute Error:** The mean absolute error is a common measure of forecast error in time series analysis, where the terms "mean absolute deviation" is sometimes used in confusion with the more standard definition of mean absolute deviation. The same confusion exists more generally..
- **Root Mean Square Error:** The root-mean-square deviation (RMSD) or root-mean-square error (RMSE) is a frequently used measure of the differences between values predicted by a model or an estimator and the values actually observed. **Classified Rate:** It is defined as the classified samples to the total number of samples.

## 7.2 Experimental Dataset

The log data set which includes various anomalies found in the log files. The entire dataset contains the two class of anomalies form in the ATLAS appliance software when trying to update the patch.

### 7.2.1 Training data

From the Table 8.1 represents the training set from the issues faced during updating the ATLAS appliance software. The data is tabulated from the view result and each data sample is labelled with the class.

**Table 8.1: Training data**

| Time Frame | Level | Validation | Error  | Class   |
|------------|-------|------------|--|---------|
| 00 to 08   | Error | No         | bin is corrupted                               | WA_1000 |
| 00 to 08   | Warn  | No         | bin is corrupted                               | WA_1000 |
| 00 to 08   | Error | No         | Memory full                                    | WA_1001 |
| 09 to 16   | Error | No         | bin does not meet the pre-requisite conditions | WA_1001 |
| 16 to 00   | Error | Yes        | bin does not meet the pre-requisite conditions | WA_1001 |
| 16 to 00   | Warn  | Yes        | bin does not meet the pre-requisite conditions | WA_1000 |
| 16 to 00   | Warn  | Yes        | Memory full                                    | WA_1001 |
| 09 to 16   | Error | No         | bin is corrupted                               | WA_1000 |
| 16 to 00   | Error | Yes        | bin is corrupted                               | WA_1001 |
| 09 to 16   | Error | Yes        | bin does not meet the pre-requisite conditions | WA_1001 |
| 09 to 16   | Warn  | Yes        | bin is corrupted                               | WA_1001 |
| 09 to 16   | Warn  | No         | Memory full                                    | WA_1001 |
| 00 to 08   | Error | Yes        | Memory full                                    | WA_1001 |
| 09 to 16   | Warn  | No         | bin does not meet the pre-requisite conditions | WA_1000 |

The log file has the training data with 14 samples and 2 classes for demonstration purpose. The below table shows data used in the training set. The corresponding rules generated to derive the knowledge. The decision tree algorithm is applied to the data set in the table 8.1 the following results were derived

### 7.2.2 Classification Errors

**Table 8.8: Classification report**

|   |        |        |
|---|--------|--------|
| <b>Correctly Classified Instances</b>   | 12     | 85.71% |
| <b>Incorrectly Classified Instances</b> | 2      | 14.29% |
| <b>Kappa statistic</b>                  | 0.6889 | -      |
| <b>Mean absolute error</b>              | 0.1429 | -      |
| <b>Root mean squared error</b>          | 0.378  | -      |
| <b>Relative absolute error</b>          | 30%    | -      |
| <b>Root relative squared error</b>      | 76.61% | -      |
| <b>Total Number of Instances</b>        | 14     | -      |

From the Table 8.2 error report in the classifying the training data

### 7.2.3 Rules in DRL file

The rules generated for the following data set are shown below

```

rule"classification WA_1000"

when

LogEntry(    isError("bin is corrupted") &&

                                isValidatationDone("No") ||

                                isError("bin is corrupted") &&

                                isLevel("Error")

                                );

```

```
        then

            setClass("WA_1000");

        end

rule"classification WA_1001"

    when

        LogEntry(    isUploadError("bin_is_corrupted") &&

                    isValidationDone("Yes") ||

                    isUploadError("Memory_full") ||

                    isUploadError("bin_does_not_meet_the_pre-
                    requisite_conditions") &&

                    isLevel("Warn")

                    );

        then

            setClass("WA_1000");

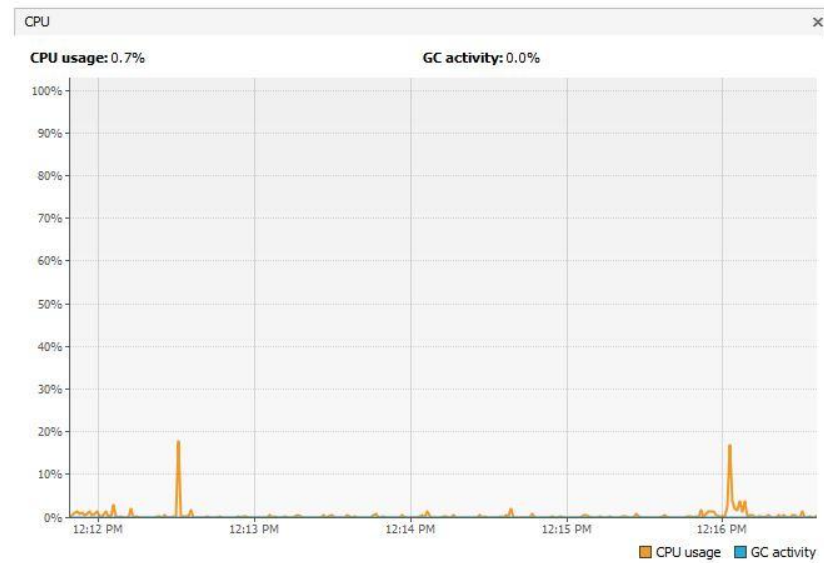
        end
```

## 7.3 Performance Analysis

The system resources utilization for the log analyzer for the log file from support dump of size 1.54GB with the extraction ratio of 0.1% are shown in the below graph.

### 7.3.1 System CPU utilization

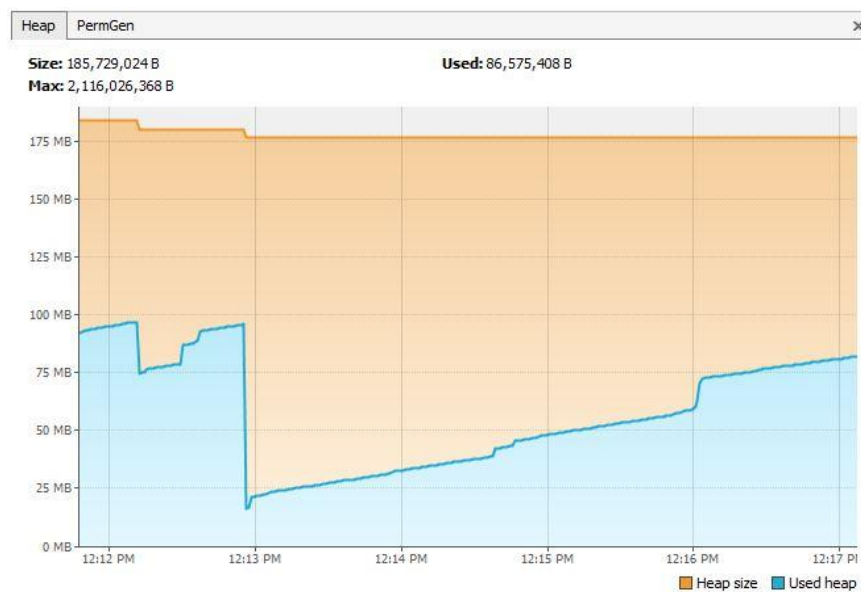
The system utilization for the log analyzer for the log file from support dump of size 1.54GB with the extraction ratio of 0.1% are shown in the figure 7.1.



**Figure 7.1 Log Analyzer CPU utilization for 1.54GB log dump**

### 7.3.2 System Memory utilization

The system utilization for the log analyzer for the log file from support dump of size 1.54GB with the extraction ratio of 0.1% are shown in the figure 7.2.



**Figure 7.2 Log Analyzer Memory utilization for 1.54GB log dump**



## 7.4 Output Result

The anomaly detected in the uploaded files is analyzed and the recommendation is provided to the user. The below screen shot shows the recommendation for the ATLAS log for the uploaded log file.

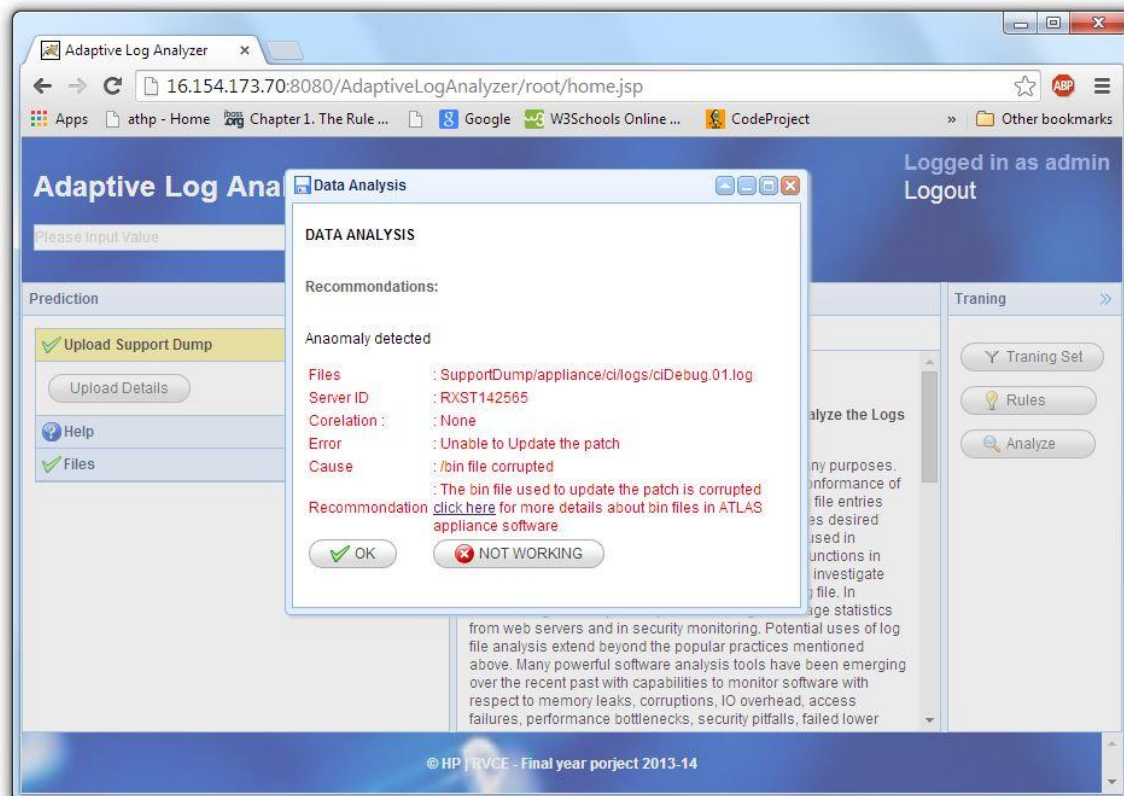


Figure 8.3 Log Analyzer CPU utilization for 1.54GB log dump

## 7.5 Inferences

The training set if classified with the classification accuracy of 85% the rules generated from the decision tree is executed on the rule engine with effective memory and CPU utilization.

## CHAPTER 8

# CONCLUSION OF KNOWLEDGE AND A RULE BASED ENGINE TO ANALYZE THE LOGS FOR TROUBLESHOOTING

A predictive software model is proposed for the classification and detection of anomalies in log. The system encompasses various machine learning and data mining techniques. The methodology begins with log collection. The collected logs are formatted and relevant data is extracted.

The results showed that adoptive log analyzer could detect the anomalies patrons and classify the given log file appropriately to an accuracy of 85.12%. An appropriate recommendation module also has been developed to provide recommendation to troubleshoot the problem.

### 8.1 Limitation of the Project

- a. The current log analyzer address only limited number of log formats.
- b. The universal standard has to be followed in logging the application.
- c. Log collections are done manually.
- d. The system is not addressed on scalability issue.

### 8.2 Future Enhancements

The comprehensive proposed approach can be further enhanced with the following functionalities:

- a. The system was built for Tomcat container and could handle logs of smaller size. The current setup thought to be upgraded and deployed on hadoop cluster which can be used to leverage the map reduction technique as a part of future work.
- b. This can make the log analyzer highly scalable. Also, multiple logs can be analyzed in parallel.
- c. The issues can be auto healed but running the automation scripts.

---

## REFERENCES

- [1] Dileepa Jayathilake "Towards Structured Log Analysis" *International Joint Conference on Computer Science and Software Engineering* IEEE pp 259 – 264, 2012.
- [2] Dileepa Jayathilake. "A Novel Mind Map Based Approach for Log Data Extraction" *International Conference on Software and Computer Applications IPCSIT* pp 285 – 311, 2011 .
- [3] Anton Chuvakin, Kevin Schmidt, Chris Phillips. "Logging and log management the authoritative guide to understanding the concepts surrounding logging and log management" Publication – Elsevier, Editin-2, 2013.
- [4] Nathaphon Kiatwonghong, Songrit Maneewongvatana. "Intelli-Log : A Real-time Log Analyzer" *2nd International Conference on Education Technology and Compute* IEEE pp 158-120, 2010.
- [5] Fu, Q., Lou, J.G., Wang, Y., and Li, J.,. "Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis" *3rd International Conference on Education Technology and Compute* IEEE ICDM, pp. 149 – 158, 2009.
- [6] Paul Browne "JBoss Drools Business Rules" Publication – PACKT, Edition - 2 2009.
- [7] Hellerstein, J., Ma, S., and Perng, C. "Discovering Actionable Patterns in Event Data" *IBM System Journal*, vol. 41(3) pp 200-253, 2002.
- [8] Jiang, Z., Hassan, A., Hamann, G, and Flora, P.,An "Automated Approach for Abstracting Execution Logs to Execution Events" *Journal of Software Maintenance and Evolution: Research and practice* vol. 20, pp. 249–267, 2008.
- [9] Tao P., Wanli Z., "Data Mining for Network Intrusion Detection System in Real Time",*International Journal of Computer Science and Network Security*, , Vol.6 pp 145-150, 2006.

- 
- [10] Stuart Jonathan Russell, Peter Norvig. "Artificial Intelligence: A Modern Approach" Publication - Prentice Hall, Edition – 3, 2010.
  - [11] Jianwen Wei, Yusu Zhao, Kaida Jiang, RuiXie,Yaohui Jin. "Analysis farm: A cloud-based scalable aggregation and query platform for network log analysis" *Cloud and Service Computing (CSC), International Conference on Digital Object Identifier* Pages: 354 – 359, 2011.
  - [12] Nor Badrul Anuar, Hasimi Sallehudin, Abdullah Gani, Omar Zakari, "Identifying False Alarm For Network Intrusion Detection System Using Hybrid Data Mining And Decision Tree" *Malaysian Journal of Computer Science*, Vol. 21(2), Pages 105 – 140, 2008.
  - [13] MahendraPratap Yadav, Pankaj Kumar Keserwani, Shefalika Ghosh Samaddar, "An Efficient Web Mining Algorithm for Web Log Analysis: E-Web Miner" *1st Int'l Conf. on Recent Advances in Information Technology*, vol 2 pp 140-150, 2012 .
  - [14] Gabor Kiss "Using datamining tools in analyzing undergraduate paper results in Computer Science at Óbuda University" *International Conference on Digital Object Identifier* pp 140-145, 2010.
  - [15] E. Agichtein and V. Ganti. "Mining reference tables for automatic text segmentation". *10<sup>th</sup> International Conference on Knowledge Discovery and Data Mining*, vol 2 pp 120-130, 2004.
  - [16] Daniela S, Christopher J. Hinde and Roger G. Stone "Naïve Bayes vs. Decision Trees vs. Neural Networks in the Classification of Training Web Pages" *International Journal of Computer Science Issues*, Vol. 4, No. 1, pp 135-145, 2009.
  - [17] Dimitris Kolovos, Louis Rose, Antonio García-Domínguez, Richard Paige "The Epsilon Book" Publication- Eclipse.org Edition – 2 2011.
  - [18] Wenke L., Salvatore I. S. , Philip K. C., Eleazar E. , Wei F., Matthew M., Shlomo H. , and Junxin Z., "Real Time Data Mining-based Intrusion Detection", *Computer Science Department, Columbia University*. Vol 2 pp 140-145, 2011.
-

- 
- [19] Tao P., Wanli Z., 2006, "Data Mining for Network Intrusion Detection System in Real Time", *International Journal of Computer Science and Network Security*, Vol.6 pp 130-135, 2008.
  - [20] J. Valdman. "Log file analysis. Technical Report" *Department of Computer Science and Engineering*, pp – 120-130, 2001.
  - [21] J. H. Andrews. "Theory and practice of log file analysis". *Technical Report 524, Department of Computer Science, University of Western Ontario*, pp 150-154, 1998.
  - [22] Chuck Lam. "Hadoop in Action" Publication - Manning Publications Company, Edition – 1, 2011.
  - [23] Tong, Wang, Pi-lian He, "Web Log Mining by an Improved Apriori" *All Algorithm World Academy of Science, Engineering and Technology*, Vol 4, pp 97-100, 2005.
  - [24] Dunham., Margaret H., "Data Mining Introductory and Advanced Topics" *Beijing: Tsinghua University Press*, p195-220, 2003.
  - [25] Han Jiawei, Kamber Micheline "Data Mining Concepts and Techniques". *Beijing: China Machine Press*, pp 290-297, 2003.
  - [26] Bain Tony "SQL Server 2000 Data Warehouse and Analysis Services". *Beijing: China Electric Power Press*, pp 443-470, 2003.
  - [27] Bin, Lin jie, de, Liu ming and xiang, Chen "Data mining and OLAP Theory & Practice". *Beijing: Tsinghua University Press*, pp 194-244, 2003.
  - [28] Tong, Wang and Pi-lian , He, "Web Log Mining by an Improved Apriori" *All Algorithm World Academy of Science, Engineering and Technology*, Vol 4, pp 97-100, 2005.
  - [29] Wen-Hai Gao "Research On Client Behaviour Pattern Recognition System Based On Web Log Mining", *Ninth International Conference on Machine Learning and Cybernetics, Qingdao, IEEE*, pp 466-470, 2010.
-

- 
- [30] Levi Albert, Kocetin Kaya, Cvenient “Secure Electronic Payment Protocol Based on”, *The 17th Annual Computer Security Applications Conference*, Vol 4, pages 286-295, 2001.
  - [31] Guo, Di, “Collector Engine System: A Web Mining Tool for ECommerce, Proceedings of the First International Conference on Innovative Computing, Information and Control”. *2nd International Conference on Computer Engineering and Technology* Vol 4, pp 124-135, 2010.
  - [32] Mei Li and Cheng Feng, “Overview of WEB Mining Technology and Its Application in E-commerce” *2nd International Conference on Computer Engineering and Technology* , IEEE pp 277-280, 2010.
  - [33] Yue ,Chuan, Xie,Mengjun, and Wang, Haining, “Automatic Cookie Usage Setting with CookiePicker”, *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, IEEE pp 54 – 80, 2007.
  - [34] Pitkow, J. “In Search of Reliable Usage Data on the WWW”. *In Proceedings of the Sixth international WWW Conference*, pp 124-135, 2010.
  - [35] Ding Xiao, Yi Tong, Haitao Yang, Mudan Cao. “The Improvement for Rete Algorithm”. *International Conference on Software Engineering*, pp 145-155, 2009.
  - [36] Zhang Yang-Qing, Lu Xiao-Jun. “The operation consistency study for Rijndael algorithm and Reed-Solomon algorithm in Galois field for ICACIA” *2nd International Conference on Computer Engineering and Technology* Vol 4, pp 124-135, 2009
  - [37] Gu Xiao-Feng, Liu Lin. “Data classification based on artificial neural networks” *ICACIA*, pp 120-130, 2008.
  - [38] J. H. Andrews, "Testing using log file analysis: tools, methods and issues," Publication – Proc, Edition – 4, 2013.

- 
- [39] T. Takada and H. Koike, "Mielog: a highly interactive visual web browser using information visualization and statistical analysis," *Conf. on System Administration*, pp. 133-144, 2002.
  - [40] J. Valdman, "Log file analysis," *Department of Computer Science and Engineering*, pp 145-155, 2001.
  - [41] J. H. Andrews, "Theory and practice of log file analysis," *Department of Computer Science, University of Western Ontario., Tech. Rep.*, pp 142-160, 1998.
  - [42] T. Buzan and B. Buzan, "The Mind Map Book". Publication - Penguin Books, Edition – 1, 1994.
  - [43] J. Cowie and W. Lehnert, "Information extraction," *Comm. ACM* , pp. 80-91, 1996.
  - [44] J. Abela and T. Debeaupuis, "Universal Format for Logger Messages," *The Internet Engineering Task Force IEEE*, vol.21(2), pp 114 – 140, 2013.
  - [45] E. Agichtein and V. Ganti."Mining reference tables for automatic text segmentation". *Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle A*, pp145-155, 2004.
  - [46] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. "Adaptive name-matching in information integration". *IEEE Intelligent Systems*, vol 14 pp 125 – 136, 2003.
  - [47] V. R. Borkar, K. Deshmukh, and S. Sarawagi. "Automatic text segmentation for extracting structured records". *In Proc. ACM SIGMOD International Conf. on Management of Data, Santa Barabara,USA*, pp 147-150, 2001.
  - [48] Borthwick, J. Sterling, E. Agichtein, and R. Grishman. "Exploiting diverse knowledge sources via maximum entropy in named entity recognition". *In Sixth Workshop on Very Large Corpora New Brunswick, New Jersey. Association for Computational Linguistics.*, pp 120-125, 1998.
-

- 
- [49] Chandel P. Nagesh, and Sarawagi S. “Efficient batch top-k search for dictionary-based entity recognition. *ICDE*, pp 125-130, 2005.
  - [50] Cohen W, Ravikumar P, and S. E. Fienberg. “A comparison of string distance metrics for name-matching tasks”. In *Proceedings of the Workshop on Information Integration on the Web.*, pp 145-150, 2011.
  - [51] Lafferty J, McCallum J, Pereira F. “Conditional random fields: Probabilistic models for segmenting and labeling sequence data”. In *Proceedings of the International Conference on Machine Learning (ICML-2001)*, Williams, MA, IEEE pp 145 – 155, 2001.
  - [52] S. Lawrence, C. L. Giles, and K. Bollacker. “Digital libraries and autonomous citation indexing”. *IEEE Computer*, pp12-20, 1999.
  - [53] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large-scale optimization. *IEEE Mathematic Programming*, 45:503-528, 1989.
  - [54] R. Malouf. “A comparison of algorithms for maximum entropy parameter estimation”. In *Proceedings of The Sixth Conference on Natural Language Learning*, pages 49-55, 2004.
  - [55] McCallum and B. Wellner. “Toward conditional models of identity uncertainty with application to proper noun coreference”. In *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web*, pages 79-86, 2010.
  - [56] Parag and P. Domingos. “Multi-relational record linkage”. In *Proceedings of 3rd Workshop on Multi-Relational Data Mining at ACM SIGKDD, Seattle*, pp14-56, 2004.
  - [57] Ratnaparkhi K. “Learning to parse natural language with maximum entropy models. Machine Learning”, *IJDFRT* pp 15-48, 1999.
  - [58] Sarawagi S., and Bhamidipaty A. “Interactive deduplication using active learning”. In *Proc. of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD-2002)*, Edmonton, Canada. pp150-155, 2002.
-



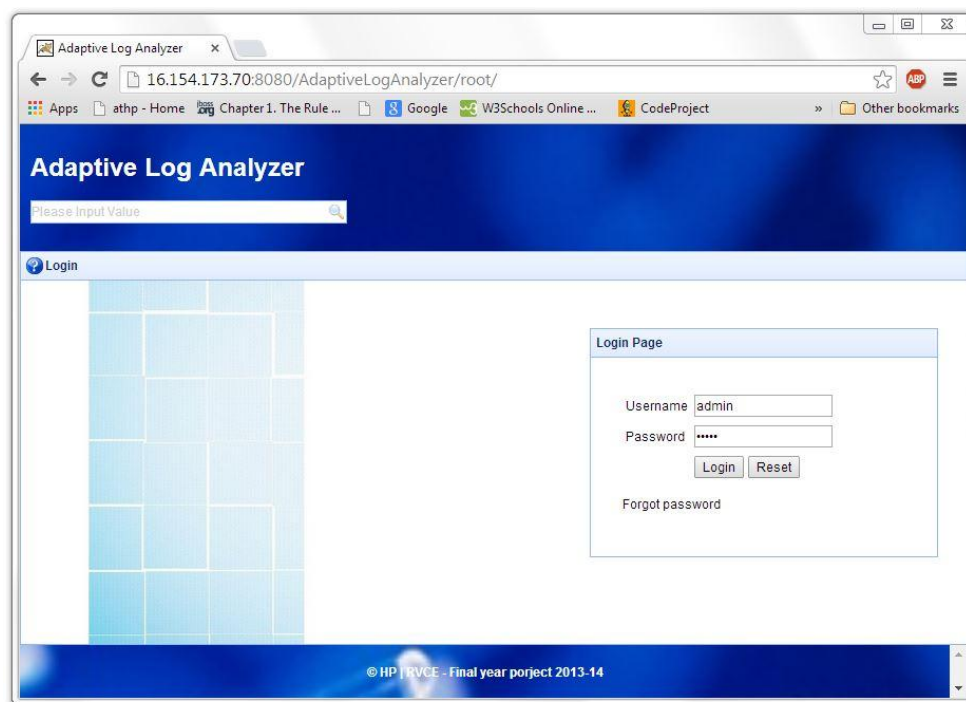
- 
- [59] Sarawagi S and Cohen W. W. “Semi-markov conditional random fields for information extraction”. *In NIPs*, pp140-155, 2004.
- [60] Seymore A, McCallum D, Rosenfeld D. “Learning Hidden Markov Model structure for information extraction”. *In Papers from the AAAI-99 Workshop on Machine Learning for Information Extraction*, pages 37-42, 1999.
- [61] Sha F and Pereira F. “Shallow parsing with conditional random fields”. *In In Proceedings of HLT-NAACL Vol. 4*, pp 150 – 175, 2013.

## APPENDIX

### APPENDIX-A

#### User Authentication

Login module is the GUI of the system, the user logs in, with user name and password. Once the user logs in the, he can access the resources of the logs. The following Figure A.1 shows the login module of adaptive log analyzer.



**Figure A.1 : Login Module for the log analyzer.**

#### Dash board

Dashboard of the system provides the user with the interface to access and to manipulate the log information and to view the contents of the logs. The dashboard also provides the access to the contents of the files once the files are uploaded. The color coding is helps in identify the log entry with probable errors. The following Figure A.2 and snapshot 3 shows the dashboard for the log analyzer.

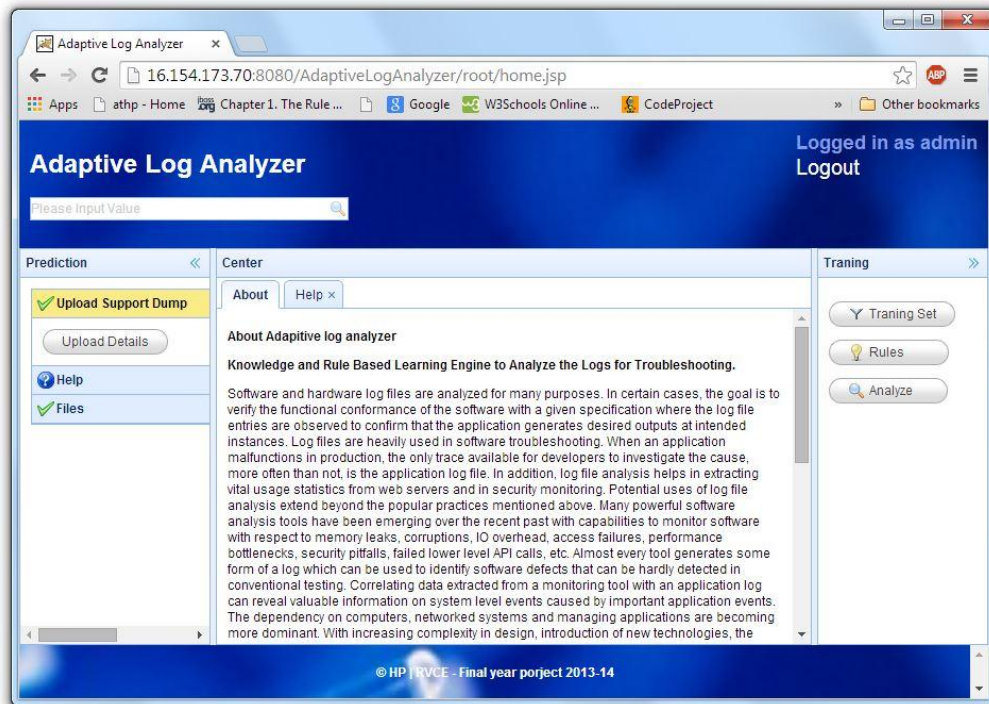


Figure A.2 : Dash board for the log analyzer

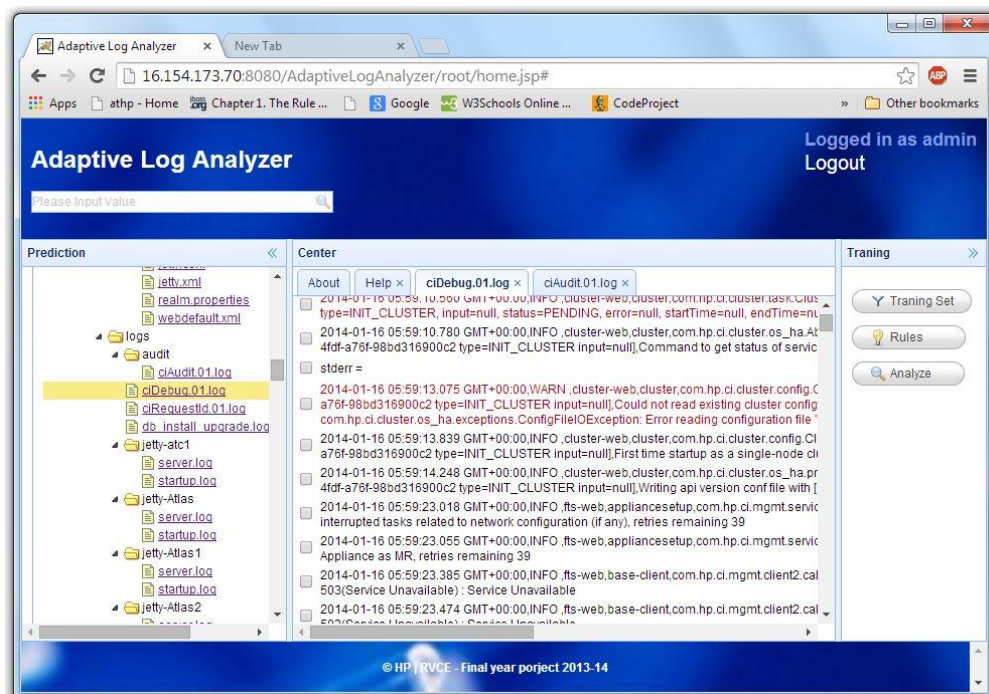
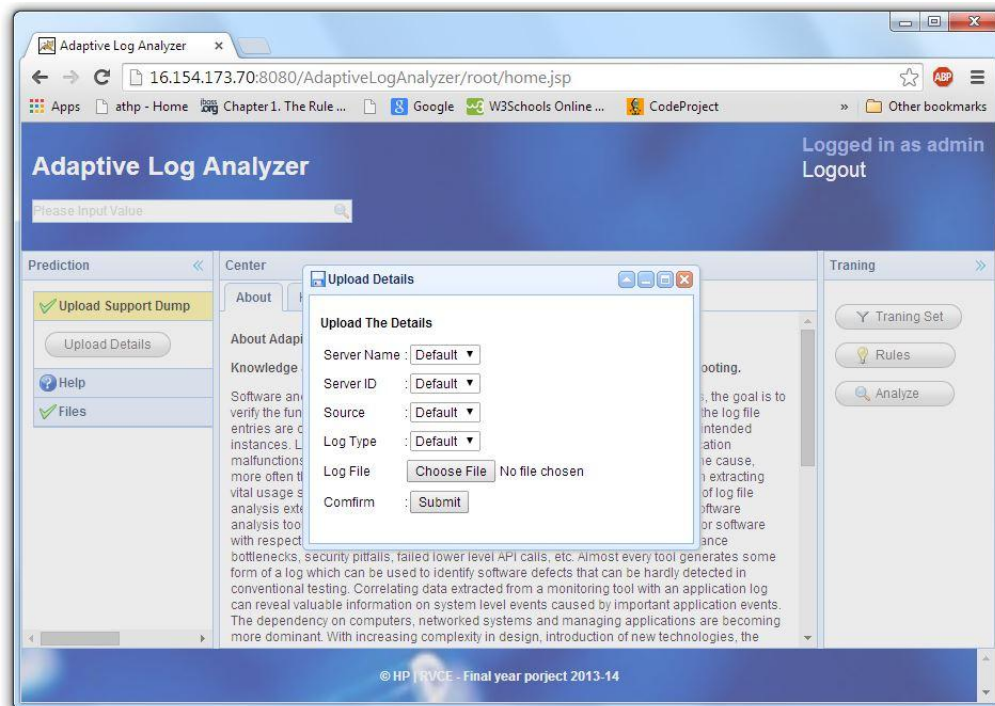


Figure A.3 : Dash board for the log analyzer with file browser

## Upload the details

The log details are uploaded by the user. The following Figure A.3 shows the uploading the details. The details include server name, server id, source component of the log, type of the log and the actual log file which will be analyzed.

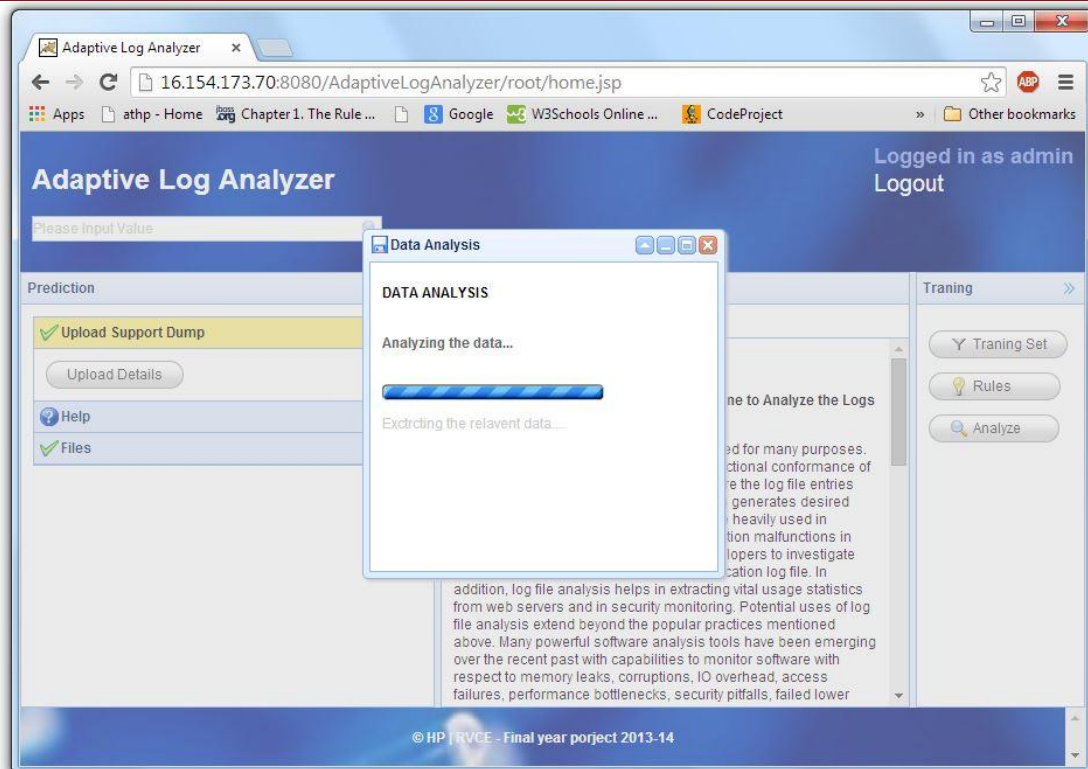


**Figure A.4 : Uploading the details for the log analyzer**

## Running the Analysis

The uploaded data is analyzed by the rule engine by firing all the rules in the working memory and validating it against the facts received from uploaded details and the probable recommendation is received. Figure A.5 and Figure A.6 shows the analysis progress and result.





Fi

Figure A.5 :Analysis progress

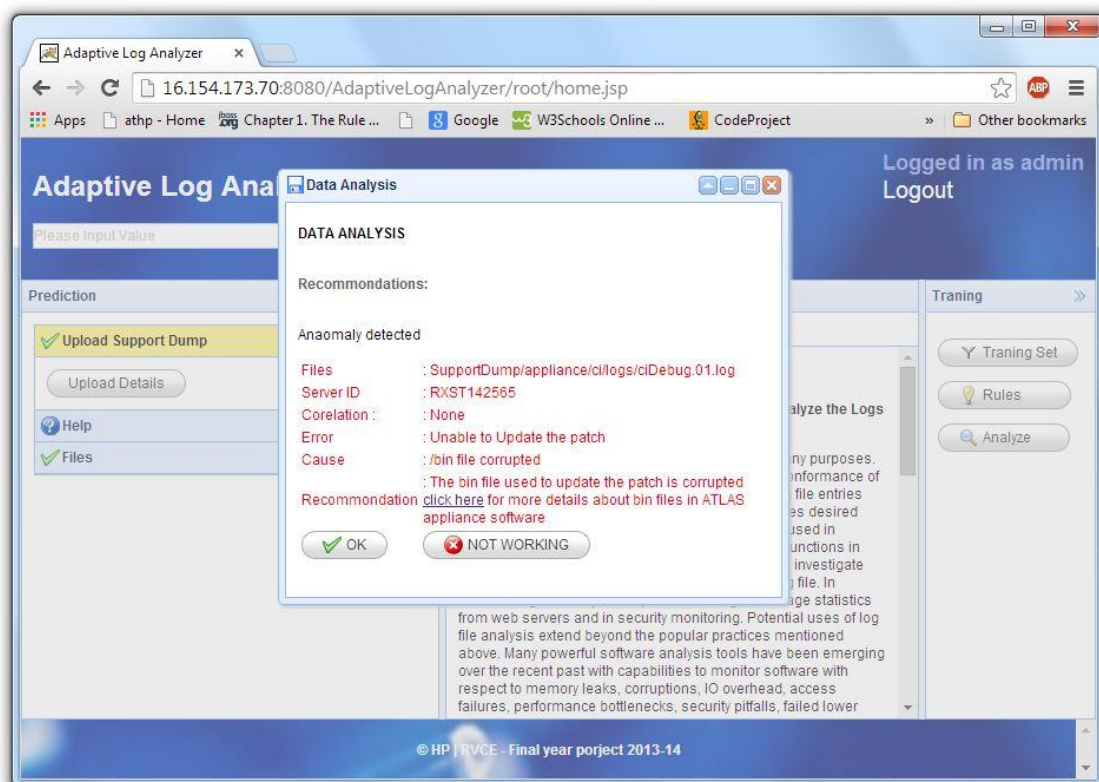


Figure A.6 :Analysis result with recommendation.

---

## APPENDIX-B

### Technical Paper published

A technical conference paper based on this project was published in International Journal for Scientific Research & Development (IJSRD Volume 2 Issue 4. ISSN: 2321-0613, pages 158-161.)

The technical paper is attached along with the appendix-B.



# Knowledge And Rule Based Learning Engine To Analyse The Logs For Troubleshooting.

Prashant Achari<sup>1</sup> Susanta Adhikary<sup>2</sup> Jayashree Madugundu<sup>3</sup> Mungara Jitendranath<sup>4</sup>

<sup>2</sup>CS Engineering, <sup>3</sup>Assistant Professor <sup>4</sup>Professor & Dean,  
<sup>1, 3, 4</sup> Dept. of ISE, R V College of Engineering, Karnataka, India  
<sup>2</sup>HP India Software Optns, India

**Abstract---** Technical support for software or hardware is always a challenge, both from the complexity and investment standpoint. Today the dependency on computers, networked systems and managing applications are becoming more dominant and only log files are available to trace the applications. Log file analysis process is heavily involved, in software, hardware as well as in network related domains. It serves for various purposes such as verifying the uniformity of the software functionality to the provided software specification, software performance check and troubleshooting the issues. 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 competency to a great deal and are labor intensive when performed manually. There is no technique available to record expert knowledge and this stands as an obstacle to automate the analysis tasks. Hence there is a need for correlating information extracted from different locations in the same log file or multiple log files further adds to this complexity. This paper describes a practical approach for analysis of the logs using supervised learning to predict and to recommend steps for troubleshooting the issue. The overall solution proposed here is to automate the analysis of logs and provide recommendations for faster response to reported problems. The log analyser self-learns the new issue and provides necessary recommendations.

**Keywords:** - Log Analysis, Supervised learning, Rule engine, Data mining.

## I. INTRODUCTION

Currently, the log analysis process has been more of manual task. The major pitfalls in the manual analysis process are that, it requires expertise, it is labour intensive, error prone and the advantage of recurrence is not used. A log contains error messages, application specific data and operational uses. In most applications the logs have different structures and different standards followed; the analysis of these logs requires tracing and finding any relevant pattern based on the reported problem in the log file. Especially for the support team the analysis of these logs becomes difficult and providing first level recommendation or troubleshooting becomes even more difficult. Apparently, the support team would need assistance from development team which incurs cost. The traditional approaches of manually analysing activities in the logs are time consuming and error prone. There is no tool that can continuously analyse and capture repetitive error occurring in the system. And there is no tool that provides recommendations for the engineer before he actually works on the issues. This will considerably increase the time invested for the analysis and troubleshooting process.

Here the challenges of analysing the existing log files using probabilistic analysis to provide aggregated information from the log files and issued alerts and recommendations based on the data present in the logs. The solution is designed with flexibility for future extension of any new generation or system log analysis with scalability in mind to process large log files.

## II. RELATED WORKS

Work has been done in type casting the patterns of the logs from single or multiple files. Frequent pattern and association rule mining for semi-structured text data has been explored by many researchers [3]. Classification and clustering techniques for mining frequent patterns have been used for troubleshooting systems and software maintenance [5]. Most of these approaches apply supervised learning based on labelled training data, which is effective when there are a limited number of message formats that can be labelled.

Efforts have been made to provide a unified infrastructure for building expert systems that automate each step involved in log file analysis which formulates means for recording the knowledge of individual experts which can be later aggregated to build an aggregated knowledgebase in the form of mind maps [6].

All of the above approaches apply value-based clustering and a frame work to capture the knowledge. We use supervised learning, decision tree algorithm to classify the logs and to generate rules from the existing training data. Rule engine provides runtime to execute these rules.

## III. METHODOLOGY

In this section, we discuss the approach for handling unstructured data in the log files, correlating the logs from multiple sources extract relevant patterns and match the patterns with the previous knowledge.

### A. Algorithms

**Classifier Algorithm:** Decision tree is constructed from a fixed set of examples. The resulting tree is used to classify future samples. The example has several attributes and belongs to a class. The leaf nodes of the decision tree contain the class name whereas a non-leaf node is a decision node. The decision node is an attribute test with each branch being a possible value of the attribute. ID3 uses information gain to help it decide which attribute goes into a decision node. The advantage of learning a decision tree is that a program, rather than a knowledge engineer, elicits knowledge from an expert.

**ReteOO:** The Rete algorithm provides the basis for a more efficient implementation rule engine. A Rete-based expert system builds a network of nodes, where each node



except the root corresponds to a pattern occurring in the condition part of a rule. The path from the root node to a leaf node defines a complete rule left-hand-side. Each node has a memory of facts which satisfy that pattern. This structure is essentially a generalized tire. As new facts are asserted or modified, they propagate along the network, causing nodes to be annotated when that fact matches that pattern. When a fact or combination of facts causes all of the patterns for a given rule to be satisfied, a leaf node is reached and the corresponding rule is triggered.

### B. Architecture overview

Figure 1 shows the architectural elements of our solution. The design is based on four major components. (1) The interpretation of the information of interest from both binary and text with an arbitrary structure and format. (2) Analytics Engine which calculates the uncertainty of each of the attributes in the knowledge base, using entropy, to form a decision tree and derives rules for information extraction from the formatted log. (3) Rule Engine provides runtime environment for validating the rules generated by the Analytics Engine against the formatted data. (4) Knowledge Base is the labelled training data.

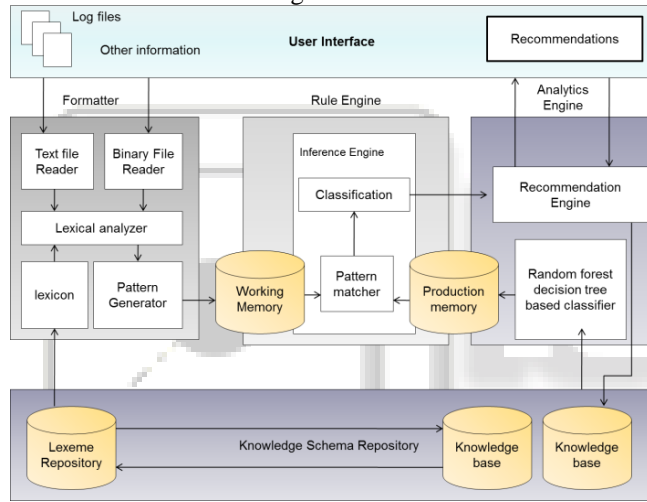


Fig. 1: Architecture elements

The first architecture element basically shows the data formatter engine which will process the unstructured or semi structured logs from the support dump to a universal format. The logs from various sources and platform have different file formats and also there is no standard followed in logging, which make it difficult in analyzing the logs. Formatter consists of lexical analyzer which searches for specific patterns available in the knowledge base and produces a structured data. The each attribute in the knowledge representation matrix is assigned with specific value from the knowledge base; the attributes are parameters like time stamp, source and destination IP for a network related log and source component, session id and other related information for application log. Regular expression is used to match the pattern.

The second architecture element shows the analytics engine which uses supervised learning to classify the log information using decision tree algorithm, sufficient set of training data is provided in the initial state. When the system is in training phase the engineer manually goes through the log file and creates an analogues structured data along with the label or classification, representing the issue

and corresponding recommendation for the issue. Every time the log is analysed manually it is captured and represented in the knowledge base in the form of a knowledge representation matrix. Once sufficient data is available in the knowledge base, the label, representing the type of issue and corresponding recommendation for resolution, is classified using the decision tree algorithm. The best predictor forms the root node of the tree. The position of the nodes in the tree structure is decided by calculation the information gain of each of the attributes in the knowledge base; one with the highest information gain will be the root node. The rest of the nodes are computed by recursively calling decision tree. Information gain is obtained using the entropy of the attribute.

Entropy  $H(S)$  is a measure of the amount of uncertainty in the (data) set  $S$

$$H(S) = - \sum_{x \in X} P(x) \log_2 P(x)$$

Where,

$S$  - The current (data) set for which entropy is being calculated

$X$  - Set of classes in  $S$

$P(x)$  - The proportion of the number of elements in class  $x$  to the number of elements in set  $S$

Information gain  $IG(A)$  is the measure of the difference in entropy from before to after the set  $S$  is split on an attribute  $A$ .

$$IG(A) = H(S) - \sum_{t \in T} P(t)H(t)$$

$H(S)$  - Entropy of set  $S$

$T$  - The subsets created from splitting set  $S$  by attribute  $A$  such that  $S = \bigcup_{t \in T} t$

$P(t)$  - The proportion of the number of elements in  $t$  to the number of elements in set  $S$

$H(t)$  - Entropy of subset  $t$

| $a_1$    | $a_2$    | $a_3$ | ... | $a_n$    | $C$   |
|----------|----------|-------|-----|----------|-------|
| $v_{11}$ | $v_{21}$ | ...   | ... | $v_{i1}$ | $c_1$ |
| $v_{12}$ | $v_{22}$ | ...   | ... |          |       |
| ...      | ...      |       |     | $v_{ij}$ | $c_m$ |

Where

$a_1, a_2, a_3, \dots a_n \in A$  Attributes

$v_{11}, v_{12}, v_{13} \dots v_{ij} \in V$  Values

$c_1, c_2, c_3 \dots c_n \in C$  Classification

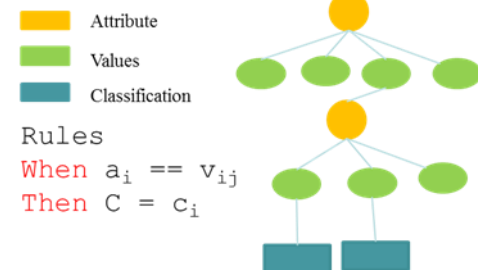


Fig. 2: Training data and decision tree

The knowledge is represented in the form of rules that are validated against the test data. The third architecture element is the rule engine, provides a runtime environment for firing the rules. The rules are derived by decision tree. The rules are both human as well as machine readable. Knowledge representation schema is the labelled data with



the structured entry for log file. The values and attributes serve as the tokens and lexeme in the lexical analyser.

### C. Experimental setup

We prepared the prototype with java programming language along with JBoss Drools rule engine. We selected java because it had drools rule engine in java and was easy to write with many support functions. For rule engines, we selected Drools rule engine because it is one of best free rule engines with high performance. We needed to examine the algorithm. Then we decided to select some type of log to process, which reduce implementation time. The select logs to process were only web server log, DHCP log and radius log. However, the algorithm could support many type of computer logs with filtering support options.

The logs from multiple sources are correlated to and aggregated into generic format with all the relevant data from the previous knowledge. The below figure 3 shows the correlating the data from the multiple sources.

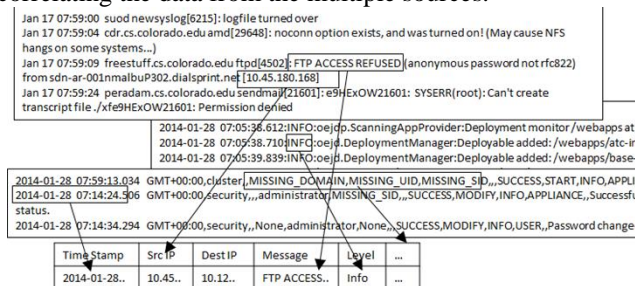


Fig. 3: Correlating and aggregating the logs.

The log from multiple sources where correlated to form universal format which was used to validate against the available knowledge. The below table shows the training data from the multiple log sources and correlated to a common format along with the label, representing which class it belongs. The decision tree algorithm is applied to classify the training data and the resulting classification tree represents the knowledge derived. The below figure shows the training data collected from correlating web server log, DHCP log and radius logs. And these are classified into two class.

| Level | IP Address   | Authentication | Upload Error      | Classification |
|-------|--------------|----------------|-------------------|----------------|
| Error | 10.63.140.37 | Failed         | permission denied | WA_1000        |
| Warn  | 10.63.140.37 | Failed         | permission denied | WA_1000        |
| Error | 10.63.140.37 | Failed         | Memory full       | WA_1001        |
| Error | 10.45.14.25  | Failed         | TP Access Refuse  | WA_1001        |
| Error | 10.75.24.63  | SUCCESS        | TP Access Refuse  | WA_1001        |
| Warn  | 10.75.24.63  | SUCCESS        | TP Access Refuse  | WA_1000        |
| Warn  | 10.75.24.63  | SUCCESS        | Memory full       | WA_1001        |
| Error | 10.45.14.25  | Failed         | permission denied | WA_1000        |
| Error | 10.75.24.63  | SUCCESS        | permission denied | WA_1001        |
| Error | 10.45.14.25  | SUCCESS        | TP Access Refuse  | WA_1001        |
| Warn  | 10.45.14.25  | SUCCESS        | permission denied | WA_1001        |
| Warn  | 10.45.14.25  | Failed         | Memory full       | WA_1001        |
| Error | 10.63.140.37 | SUCCESS        | Memory full       | WA_1001        |
| Warn  | 10.45.14.25  | Failed         | TP Access Refuse  | WA_1000        |

Fig. 4: Sample training data in universal format

The derived knowledge from the decision tree is used to generate the rules the below figure shows rules generated for Drools rule engine that are validated against the new logs.

```

rule "classification WA_1000"
when
    LogEntry(
        isError("permission denied") &&
        isAuthentication("Failed") ||
        isError("FTP Access Refused") &&
        isLevel("Error")
    );
then
    setClass("WA_1000");
end

rule "classification WA_1001"
when
    LogEntry( isUploadError("permission denied") &&
        isAuthentication("SUCCESS") ||
        isUploadError("Memory_full") ||
        isUploadError("FTP Access Refused") &&
        isLevel("Warn"));
then
    setClass("WA_1000");
end

```

Fig. 5: Sample training data in universal format

The figure below shows self-explanatory sequence diagram for the log analyser.

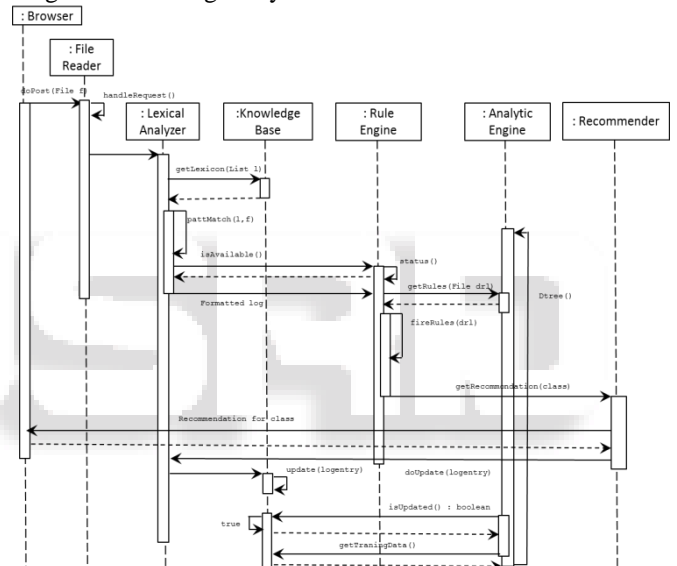


Fig. 6: Sequence diagram

## IV. RESULT AND FUTURE WORK

The proposal of adaptive log analyser based on rule engines is successfully completed and validated for various log file formats. The decision tree perfectly classifies the training data which is used by the rule engine. The recommendations given by the system is observed to give Maximum percentage of correct results.

The system was built for Tomcat container and could handle logs of smaller size. The current setup thought to be upgraded and deployed on hadoop cluster which can be used to leverage the map reduction technique as a part of future work. This can make the log analyser highly scalable. Also, Multiple logs can be analysed in parallel.

## REFERENCES

- [1] Nathaphon Kiatwonghong, Songrit Maneewongvatana. 2010 2nd International Conference on Education Technology and Compute. Intelli-Log : A Real-time Log Analyzer.
- [2] CAPRI: type-CAsted Pattern and Rule mIner. 2013. At: <http://research.cs.queensu.ca/home/farhana/capri.html>.

- [3] Fu, Q., Lou, J.G., Wang, Y., and Li, J., 2009. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. In proc. of the IEEE ICDM, pp. 149 – 158, Miami, FL.
- [4] Hellerstein, J., Ma, S., and Perng, C., 2002. Discovering Actionable Patterns in Event Data. IBM System Journal, vol. 41(3).
- [5] Jiang, Z., Hassan, A., Hamann, G, and Flora, P., 2008. An Automated Approach for Abstracting Execution Logs to Execution Events. Journal of Software Maintenance and Evolution: Research and Practice, vol. 20, pp. 249–267.
- [6] Dileepa Jayathilake. 2011 International Conference on Software and Computer Applications IPCSIT vol.9, Singapore.

