Agile Business Suite Log Analysis and Stochastic Modeling of Transactions

Anindya Mukherjea

Department of Computer Science and Engineering
National Institute of Technology Rourkela

Agile Business Suite Log Analysis and Stochastic Modeling of Transactions

Thesis submitted in partial fulfillment

of the requirements of the degree of

Master of Technology

in

Computer Science and Engineering
(*Specialization*: *Computer Science and Engineering*)

by

Anindya Mukherjea

(Roll Number: 214CS1140)

based on research carried out

under the supervision of

Prof. Korra Sathya Babu

May, 2016

Department of Computer Science and Engineering
National Institute of Technology Rourkela

Department of Computer Science and Engineering
National Institute of Technology Rourkela

May 20, 2016

Certificate of Examination

Roll Number: *214CS1140*
Name: *Anindya Mukherjea*
Title of Dissertation: *Agile Business Suite Log Analysis and Stochastic Modeling of Transactions*

We the below signed, after checking the dissertation mentioned above and the official record book (s) of the student, hereby state our approval of the dissertation submitted in partial fulfillment of the requirements of the degree of *Master of Technology* in *Computer Science and Engineering* at *National Institute of Technology Rourkela*. We are satisfied with the volume, quality, correctness, and originality of the work.

Prof. Korra Sathya Babu
Principal Supervisor

External Examiner

Prof. Santanu Kumar Rath
Head of the Department

Department of Computer Science and Engineering
National Institute of Technology Rourkela

Prof. Prof. Korra Sathya Babu
Professor

May 20, 2016

Supervisor's Certificate

This is to certify that the work presented in the dissertation entitled *Agile Business Suite Log Analysis and Stochastic Modeling of Transactions* submitted by *Anindya Mukherjea*, Roll Number 214CS1140, is a record of original research carried out by him under my supervision and guidance in partial fulfillment of the requirements of the degree of *Master of Technology* in *Computer Science and Engineering*. Neither this thesis nor any part of it has been submitted earlier for any degree or diploma to any institute or university in India or abroad.

_____
Prof. Korra Sathya Babu

Dedication

Dedicated to my parents and teachers

Signature

Declaration of Originality

I, *Anindya Mukherjea*, Roll Number *214CS1140* hereby declare that this dissertation entitled *Agile Business Suite Log Analysis and Stochastic Modeling of Transactions* presents my original work carried out as a postgraduate student of NIT Rourkela and, to the best of my knowledge, contains no material previously published or written by another person, nor any material presented by me for the award of any degree or diploma of NIT Rourkela or any other institution. Any contribution made to this research by others, with whom I have worked at NIT Rourkela or elsewhere, is explicitly acknowledged in the dissertation. Works of other authors cited in this dissertation have been duly acknowledged under the sections "Reference" or "Bibliography". I have also submitted my original research records to the scrutiny committee for evaluation of my dissertation.

I am fully aware that in case of any non-compliance detected in future, the Senate of NIT Rourkela may withdraw the degree awarded to me on the basis of the present dissertation.

May 20, 2016
NIT Rourkela

Anindya Mukherjea

Acknowledgment

I would like to thank my project guide Prof. Korra Sathya Babu at National Institute of Technology, Rourkela for his domain knowledge and timely advice.

Secondly, I would like to thank the entire Agile Business Suite team at Unisys Corporation, ATCI, Bangalore for their immense help in the project starting from accumulation of required log files to clarification of domain specific doubts and concerns. In particular, I would like to acknowledge the support given to me by Mr. Venkitaraman Anatharama and Mr. Karthikeyan Rajagopalan.

Last but not least, my sincere thanks goes to my family and friends who have supported me during the entire course of this project.

April 20, 2016
NIT Rourkela

Anindya Mukherjea
Roll Number: 214CS1140

Abstract

Agile Business Suite (ABSuite) is an application development and deployment product that can define, generate and manage complete, highly scalable, real world, platform independent applications. The product was developed by Unisys Corporation with the intention to allow developers to rapidly develop their complete applications without having to think about low level platform dependent implementation details thus saving them from having to write thousands of lines of code. Many enterprises like Banks, Healthcare Facilities, Traffic Management, Financial Institutions and etc. use ABSuite to develop their applications and automate their business logic.

Once deployed on the application servers, the generated applications are managed by the ABSuite Runtime framework. One important function of the ABSuite Runtime is to maintain a set of log files that contain immense information on system behavior, communication between the system and the runtime framework and user interaction with the system. There are various kinds of log files that are generated by the ABSuite Runtime, each having its own format and vast amounts of hidden knowledge stored in them. These log files contain important information regarding usage patterns, system failure patterns and other performance bottlenecks. Thus proper analysis of these log files is necessary to obtain vital information which will help in optimizing system performance, easing maintenance tasks, identifying hidden bugs and abnormal behavior and adopting better design strategies.

Log files are produced by almost all devices, systems and protocols. In general, the analysis of any kind of log file is not an easy task. There are several challenges that need to be addressed first in order to obtain proper and accurate results. The first issue is directly related to the fact that ABSuite log files are often huge and heterogeneous in nature. Standard Algorithms fail in such conditions and thus the task of log analysis requires a different approach. Secondly, identification of required information and the log files storing such information requires domain specific knowledge and expertise regarding the generated log files. Without prior information on what to look for and where to look for, the analysis process will become impossible. Last but not least is the issue of unstandardized format of each of the ABSuite log files which makes processing a difficult task. Keeping the above things in mind, this thesis presents some of the ideas behind developing a tool that automatically analyzes the generated ABSuite log files and extracts information that will help the developers to optimize the application's performance and reduce the system's downtime and maintenance cost.

In this thesis we discuss three stages of analysis. The first stage (Ispec Analysis) finds basic statistical parameters like trigger count, frequency and probability distribution of each transaction occurring in the system. This information is used in subsequent stages to obtain more accurate results and hypothesis. In addition, We also present a way to use the results of this stage to estimate usage and traffic patterns, peak loading conditions and heavily used modules of the system. The second stage (Response Time Analysis) obtains mean time to response for each transaction. This information is used to identify transactions with high latencies and find the most likely cause of those latencies. The third stage (Exception Analysis) finds which transactions frequently generate exceptions and how many types of exceptions are generated by each transaction. The results of this stage help in identifying the number, type, severity and cause of exceptions generated by the system. The results of this stage can also help in taking high level decisions like whether system resources need to be increased to avoid deadlocks, whether there is any need to increase bandwidth allocation, whether there is any need to change the current design of the system and etc.

Finally, we present and compare the analysis results of all the three stages for two sample applications (developed using ABSuite) with known usage characteristics. We also explain how the developers can use the results of each stage of analysis to better optimize their systems using the two sample applications as case study.

***Keywords*: *Agile Business Suite*; *Log Analysis*; *Basic Ispec Analysis*; *Response Time Analysis*; *Exception Analysis*.**

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to Agile Business Suite

Enterprise Application development (EAE), developed at Unisys corporation, is one of a very select class of software products capable of generating complete applications that can be run in a very large-scale mission critical environment. Agile Business Suite (ABSuite) is a new Unisys product that builds upon the tradition of EAE and aims to further improve use of component technology as well as provide very close integration with Microsoft Windows technologies. The purpose of this change was to allow EAE systems to interact more effectively with other systems, as well as making them easier to customize and maintain. Moving to component technology also has the advantage of modernizing, and standardizing, the development environment, making it easier to attract and retain developers. ABSuite enables enterprises to develop and generate application to the Windows .NET framework. It does this by providing a distinctly different development tool as a plug-in project type within the familiar Microsoft Visual Studio .NET framework, giving customers an enhanced development environment with System Modeler and generation of complete deployment environments, including databases, application servers, and all associated artifacts.

ABSuite combines the best of the EAE toolset with concepts from object-oriented (OO) technology, offering users the richness of OO application development. For example, by building standard elements as classes and encapsulating them in a clean interface, users can customize standard applications to suit specific requirements. Specialized standard framework classes facilitate adding data or overriding behavior. The ABSuite development environment brings the ability to specify the application based on ‘what’ needs to be done rather than ‘how’ it is to be done as is inherent in most procedural OO development environments.

ABSuite enhances Visual Studio with a highly productive development environment based on high-level specification of the business model, capability to deploy complete applications, and the flexibility to easily adapt the model to evolving business needs and changes in user environments. By specifying ‘what’ rather than ‘how’, the ABSuite deployment infrastructure takes care of technology and platform changes while the developer worries only about business-level changes. With ABSuite, customers can develop

components for their applications using System Modeler, included in the ABSuite product. This development environment snaps into the Visual Studio .NET framework as a project type and provides the same look and feel as the Microsoft-supplied project types. This flexibility ensures that Agile Business Suite successfully targets the wide customer base already enjoyed by EAE as well as attracting new customers looking for an enterprise-scale development tool.

Because ABSuite is integrated into the .NET environment, users can build business solutions that include both ABSuite and other .NET components thus optimizing resource usage. An ABSuite solution may, for example, comprise ABSuite components, Visual Basic .NET, and C++ componentsand etc. Through the use of standards such as SOAP, UDDI and WSDL, these solutions may work with other clients and services generated by other IDEs such as Web Sphere Application Developer (WSAD), Visual Studio or Borland/TogetherSoft.

1.1.1 Benefits of Agile Business Suite

Agile Business Suite generates complete, customized applications capable of scaling to support hundreds and, if necessary, thousands of end users. It enables a company with limited development staff to rapidly build, deploy, and manage complex applications. With this in mind, Agile Business Suite users should enjoy the following benefits —

1. **High Productivity**
   The following features ensure high productivity for an enterprise –

   A) Decoupling of High level specification of the business model and low level implementation details.

   B) Quick and efficient Generation of the complete application from the business model.

   C) One button deployment of the application.

   D) Lifecycle management.

2. **Large Scale**
   Agile Business Suite customers will typically require a system that supports many users. The numbers of users are often in the hundreds, if not in the thousands, and potentially geographically dispersed.

3. **Ability to absorb constant change**
   Typically, an Agile Business Suite customer has an application that is subject to constant change over short periods of time. The fact that Agile Business Suite has the capability to rapidly absorb and mirror business change is a key customer benefit.

1.1.2 Model Driven Architecture

One of the greatest strengths of EAE is that it models an application at the logical level. It has allowed customers to —

1. Focus their efforts on describing their application's behavior rather than focusing on the details of implementation; and
2. Bring their applications forward from one generation of technology to another, or from one platform to another, over the life of their application.

In ABSuite, this logical model is being enhanced and simplified by merging the best of the EAE concepts with the best of the mainstream component concepts, including COM and UML, to give a more powerful development environment than either in isolation.

The design of the model for ABSuite has been based on the principle of allowing users to describe a change to the model as directly as possible, requiring the model to make whatever other changes are required as a consequence. For example, an ispec becomes output or I/O by virtue of having persistent attributes, rather than having to specify its kind and that of its attributes separately, and making sure they agree. The model is made persistent in a relational database, making changes to the model transactional, and allowing work groups to share a single model. Consequently, it ensures that all changes to the model are valid. Unlike 3GLs whose models are captured in text files, ABSuite actively prevents errors rather than simply identifying the errors later. Figure 1.1 below shows the architectural overview of an application being developed using ABSuite —

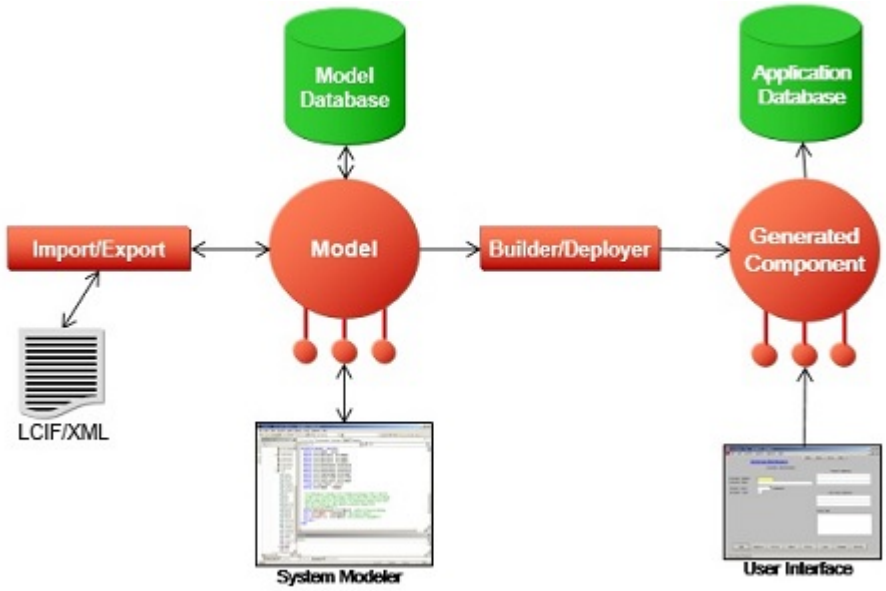


Figure 1.1: Architectural Overview of an application developed using ABSuite

1.1.3 Components of Agile Business Suite

AB Suite is made up of 2 major components —

1. ABSuite Developer ABSuite Developer includes the following –

   A) System Modeler (for modeling information systems using UML diagrams. It constitutes the Solution/App definition phase.)

   B) Builder (for generating and deploying the system developed in the Modeler. It constitutes the Solution/App generation phase)

   C) Debugger (for testing systems modeled using the Modeler).

   D) Version Control.

2. ABSuite Runtime ABSuite Runtime is installed on the target runtime platforms and provides an infrastructure in which the ABSuite components run. The ABSuite Runtime is responsible for the following tasks –

   A) Global Management

   B) DB Administration

   C) DB Audit/Recovery

   D) Transaction Management

   E) Report Management

   F) Inter System Communication

   G) Runtime Infrastructure

Figure 1.2 below shows the 2 major components of ABSuite —

1.1.4 System Modeler Development Environment

System Modeler is a model based tool for designing and developing information systems. It allows us to focus on logical requirements of a system without worrying about platform specific implementation details. Individual elements defined in the Modeler translate into multiple physical elements in the runtime system. For Example – an element representing a customer in system modeler might translate into a database table, executable code used by runtime framework and a UI. Because the modeler creates a model that is platform independent, it is much easier to change than an application created using a 3GL – one can simply update the high level definition and the Builder updates the low level implementations automatically. One can add logic to a class using system modeler's high level scripting language LDL+ or SQL. The LDL+ language includes commands for performing common data processing tasks such as value manipulation, database look ups and report control.

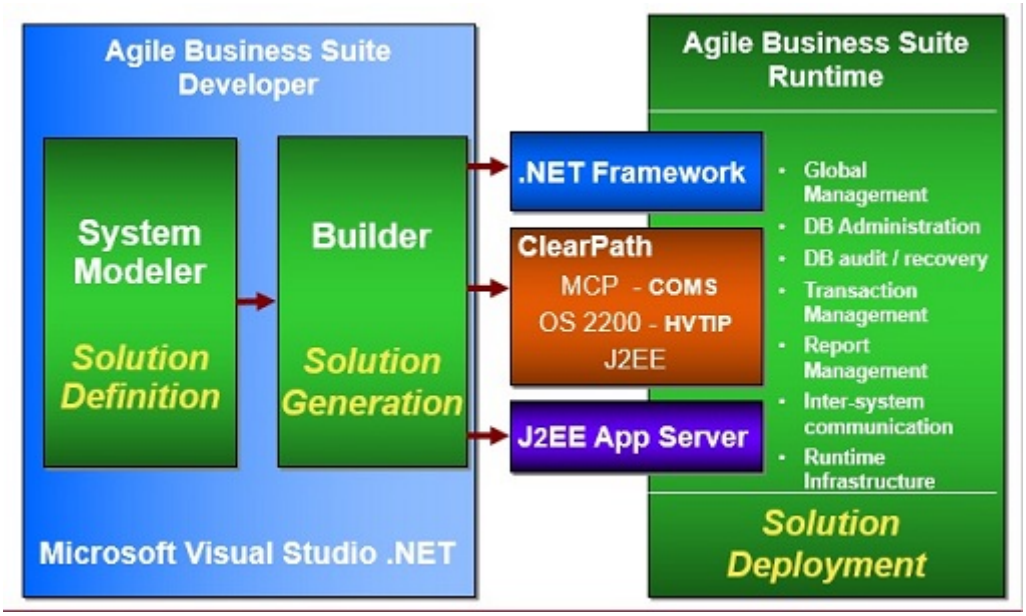


Figure 1.2: Interaction between the two major components of ABSuite

Figure 1.3 shows the System Modeler environment embedded in MS Visual Studio for developing ABSuite applications.

The System Modeler uses a number of screen panes, which include —

1. The Solution Explorer, which displays files of versionable elements of a project. Each element in the Solution Explorer can be manipulated by source control in standard ways such as addition to a version control bank, checking in, and checking out.

2. The Class View, which displays the hierarchy of classes and their members, and can be used to add new elements to the model, move and rename them. Its main function, however, is to select elements to be displayed in the Properties window of the Developer System Modeler.

3. The Properties window, which displays properties that are common to all the elements selected in the Class View or Solution Explorer. The properties are shown in table format listing the property names and values. Changes to property values are applied to all selected elements.

In addition to these views that are part of the Visual Studio environment, System Modeler also includes a Designer Window, which contains tabbed pages with different views of an element in the model. The Designer pages include —

1. Documentation, showing a WYSIWYG text editor for text or embedded OLE objects describing the element.

2. LDL+ Logic for a method, profile, or SQL script.

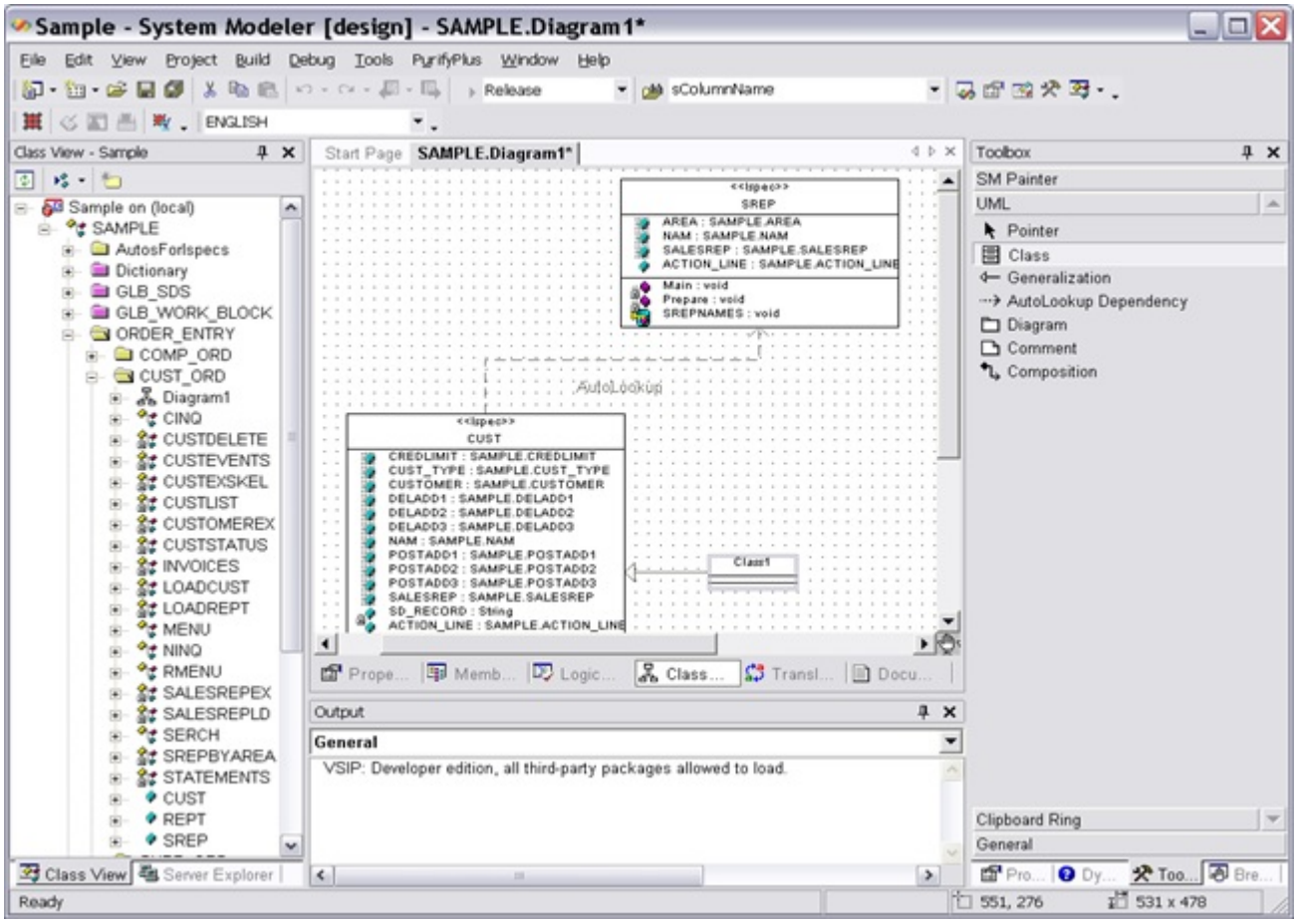


Figure 1.3: The System Modeler environment embedded in MS Visual Studio for developing ABSuite applications

3. Relationships between elements—relationships are actual, i.e. reflecting a dependency between elements.

4. Class diagram, showing a UML class diagram of an element and its classes.

5. Painter for user graphical user interfaces, reports, and teach screens.

1.1.5 Generation of Applications using ABSuite

Once the application model is developed using System Modeler, we can deploy it to one of the allowable server platforms. The paradigm and process of simple actions to build, compile, and deploy an application is well established in the ABSuite Environment. The ABSuite Builder translates design information stored in the developer model into a running database application. During this process, Builder uses input from the model, in the form of structural information, configuration information (properties), and logic. The Builder stores the latest previously generated files for each configuration. If the application has been generated previously, these files will be retrieved from the Builder cache folder instead of regenerating them.

Internal file templates are used to provide a framework for the form and structure of the various structural elements, which make up the generated application files. Typically, they are partially complete examples of the elements they represent and contain triggers that cause the Builder process to insert specific generated code fragments at particular places. File templates also provide a means of specifying code fragments to be generated in the target language when certain conditions are met. As far as possible, Builder is separated from any knowledge of the form of the generated code.

Change analysis compares the current state of the model, using the date and time of the last change, to the files comprising the previously generated application (if any) to determine which elements need to be generated. The files that comprise the application are generated along with deployment project information.

The generated C-Sharp files, which make up a C-Sharp project, are compiled and linked using the pre-compiled libraries and input to the deployment project. Following compilation and linking of the C-Sharp project, the output files are stored in the model for each configuration, to be retained for future build/change analysis of the application.

The deployment project is then built to create the deployment package (MSI). Once the deployment package has been created, the generated application files, and the output of the C-Sharp project build may be deleted and the folders containing them are removed.

1.1.6 Elements of an ABSuite Model

This section describes the elements of the ABSuite logical model of a customer's application and includes information about the following —

1. **Elements**

   All of the elements of the Agile Business Suite model share certain characteristics —

   A) Each has a name. Names are between 1 and 64 characters long, and can contain any combination of alphanumeric characters and underscores, except that they can not start with an underscore or numeral.

   B) A "Stereotype". This is a concept borrowed from UML. It is a tag that indicates to Agile Business Suite Developer how this element should be interpreted. For example, ispecs, business segments and reports are all classes in the Agile Business Suite model, but they are distinguished by their stereotype. The stereotype tells Agile Business Suite Developer that they are special kinds of classes and have special behavior in Developer and are generated differently to vanilla classes. Stereotypes provide an extensible mechanism, which is to be used in future to add new constructs

   C) A short text description.

2. **Namespaces**
   All elements, with the exception of the model itself, belong to an owning element. Elements are identified within their owner by their name, and for this reason their owner is called a "Namespace". Each kind of Namespace controls the kinds of elements it can contain. Its members might be Namespaces themselves.

3. **Classes**
   Classes are the most significant unit of an application. They describe objects and components. A Class is a specialization of a Namespace, containing other classes or Attributes, Methods, Profiles, Presentations and etc. Classes can inherit these members from their superclass. Each element in a Class can be defined to be visible —

   A) Only within its Class (private).

   B) Within its Class and any class inheriting from it (protected).

   C) Outside its Class (public).

   Business segments, ispecs, insertable GLGs, and reports become special kinds of classes, distinguished by their stereotype. There can also be classes with no stereotype, which still represent logical classes, but do not conform to any Agile Business Suite patterns. The Segment class is a component: a cohesive unit of the application that can be deployed as a unit.

   Unlike other products which implement persistent objects, ABSuite considers that all objects exist in memory only, but their persistent attributes may be read from and stored back to a record in the database at various times. A single in-memory object may load and store data belonging to a number of different database records. A common example of this is when iterating through records in the database. A single in-memory object will load its persistent attributes from successive database records.

   External classes can act as placeholders for —

   1. Ispecs in other applications (to support external auto to applications from earlier versions of Enterprise Application Environment).
   2. External components (implemented outside Agile Business Suite).

   These are each represented in the same form as other classes, but have no implementation defined (private members, logic or subsets).

Ispecs (Interface Specifications)

Ispecs (ispec-stereotyped classes) represent an entity in the business world, such as a customer, product, or vendor. Ispecs have inherent behavior related to —

1. Control of the ispec processing cycle and error handling, which includes —
   A) Initializing non-interface and non-persistent variables.
   B). Performing appropriate validation of input variables, including automatic lookups, value-checking logic, numeric validation, date validation, and ascertaining the presence of required variables.
   C) Invoking the Prepare method.
   D) Invoking the Main method.
2. Assembling error and/or output messages for transmission to the application user.

Reports

Reports (report stereotyped classes) allow one to —

1. Perform batch processing tasks. For example, we can use reports to perform bulk updates of records in the database.
2. Present raw business data stored in the database as meaningful information about the operation of the business. For example, we can use reports to produce sales receipts, invoices and sales reports. Reports are the major tool for reporting, collating and presenting information to business operation staff.
3. Consolidate data. For example, we can use reports to delete database records that are no longer needed.

4. **Attributes**

Data that belongs to a class is called an attribute. Attributes are specialized variables. Global set-up data items (GSDs) are attributes of a business segment. An ispec's data items and set-up data items (SDs) are its attributes. A report's set-up data items are its attributes. Any attribute can now be made persistent.

Ispec attributes were previously classified as input, output, IO or inquiry, which describe whether the attribute is persistent and/or appears in the user interface for input and output or output only. These characteristics are now described directly: an attribute becomes "output" by virtue of being persistent, and "input" by virtue of appearing in a user interface, or the combination of the two

5. **Methods**

Methods contain logic, which can be called by their clients. They use parameters to pass data in and/or out. They can also have local variable. The model allows

for logic to be defined in a range of different languages. In Agile Business Suite this can be LDL+ or a dialect of SQL (as used in SQL scripts). Methods are also a specialized name space, containing local variables and parameters. Performable and Callable global logic becomes a method of the business segment class, the ispec logic overrides built-in methods in the framework classes, and frames become classes of their report class with the frame layout (if any) as a presentation.

1.1.7 Transaction Processing

Transaction processing is implemented via the ABSuite segment cycle. This functionality occurs since all segment methods implicitly process transactions, and the segment cycle processes ispecs as transactions. The transaction processing cycle also determines the context in which logic commands operate. When a logic command is executed, its operating context is resolved to the stereotype of the initial class activated by the segment cycle (for ispecs and events) or the called report (for reports).

Other processing occurs within the context of the segment cycle (including the ispec cycle), such as copy cycle processing (transaction processing of copy ispecs and events), SQL script processing (of SQL scripts), and automatic entry processing.

The segment cycle is the processing cycle that occurs when the application executes an ispec or event transaction. It is controlled by the segment and defines the order in which built-in methods are called. Figure 1.4 below shows the basic segment cycle whereas Figure 1.5 shows the runtime cycle from the client's viewpoint.

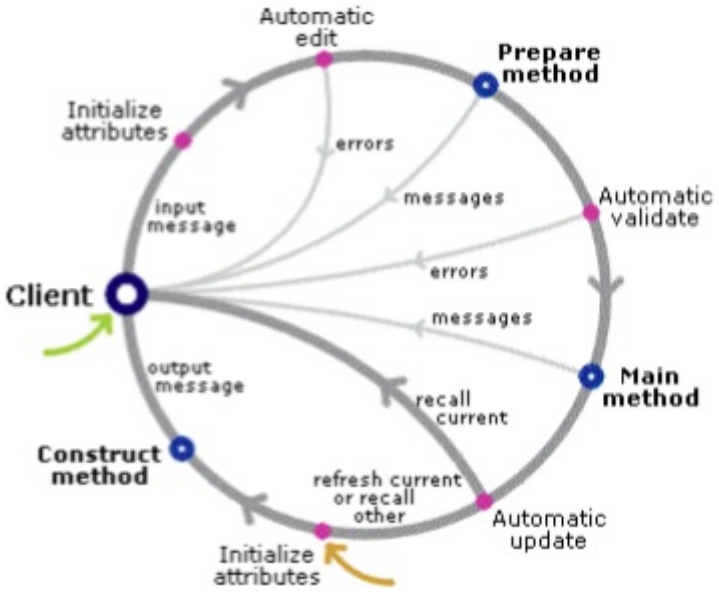


Figure 1.4: Segment Cycle (Transaction Processing Cycle)

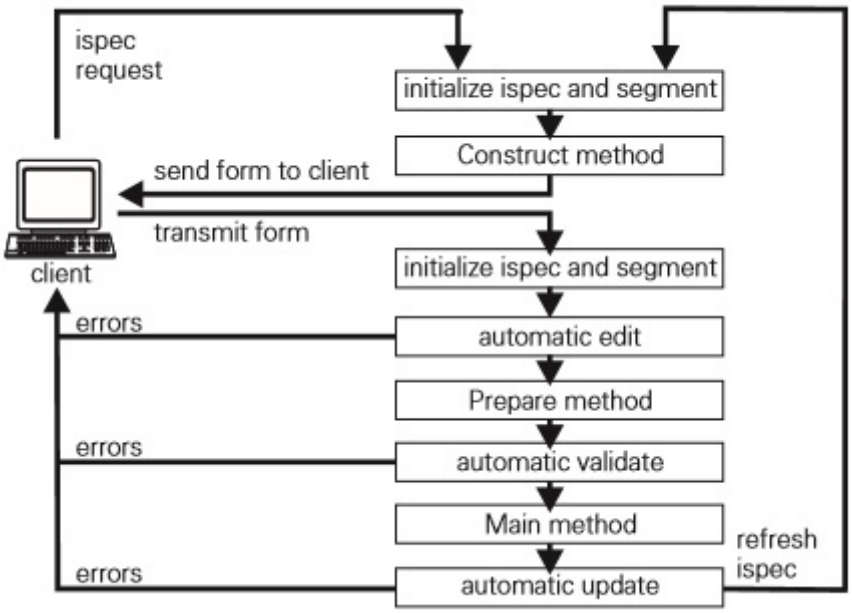


Figure 1.5: Runtime cycle (from the client's viewpoint)

Requesting an Ispec

An ispec can be requested as a result of one the following —

1. A request from the Select Ispec dialog box.
2. A Recall logic command invoked by another ispec.
3. A Recall logic command invoked by the same ispec.
4. An Abort logic command.
5. A Roc logic command..
6. A request/incoming message from an external caller via the segment's public (COM) interface..
7. An automatic refresh of the ispec..

When an ispec is requested, the following process steps occur before the ispec is ready to accept input (for ispecs with a user interface, this corresponds with its display to the application client). The orange arrow in the Figure 1.4 above indicates the starting point —

1. Segment and ispec attributes without defined initial values are initialized to their corresponding values from the input message.

2. The Construct method is called unless either the ispec is being requested due to an automatic refresh and a Message logic command has not been invoked, or the ispec has not been requested as a result of a Recall logic command invoked by the same ispec. The Construct method can be used for reasons such as the pre-filling of user interface fields, or security checking.

Transmitting an Ispec Update

An ispec update initiates the following process steps (the green arrow in the Figure 1.4 above indicates the starting point) —

1. Segment and ispec attributes without defined initial values are initialized to their corresponding values from the input message if they are in the presentation, or to the appropriate ”empty” value depending on the attribute type.

2. Automatic edit occurs – attributes with decimals are validated. Any errors are returned to the application client.

3. The Prepare method is called. The Prepare method can be used for reasons such as generating a customer number, performing any necessary validation of user input data, performing logic actions based on the user input, or recalling another ispec without processing the current ispec. Any Message or Recall logic commands invoked will halt processing at the end of the prepare method.

4. Automatic validation occurs – keys, dates, and required fields are validated; the database records corresponding to the specified keys are retrieved if they have an automatic lookup dependency. Any errors are returned to the application client.

5. The Main method is called. The Main method can be used for reasons such as checking stock-on-hand, or checking a customer credit limit for a sale. Any errors are returned to the application client.

6. Automatic update occurs – for a persistent ispec, the database record is updated (or written).

7. At this point, one of the following process steps occurs —

   A) If a Recall logic command was invoked on the same ispec, the Construct method call is skipped, and the segment cycle repeats from step 1 above.

   B) If a Recall logic command was invoked on a different ispec, the specified ispec is requested. See Requesting an ispec above for details.

   C) If the ispec’s Refresh Screen property is set to true, and no Recall logic command was invoked, the current ispec is requested. See Requesting an ispec above for details.

D) If the ispec's Refresh Screen property is set to false, and no Recall logic command was invoked, the Select Ispec dialog box is displayed. See Requesting an ispec above for details.

Transmitting an Ispec Inquiry

An ispec inquiry occurs when an ispec is transmitted with its Maint built-in presentation attribute is set to "FIR", "LAS", "NEX", "BAC", or "REC". It initiates the following process steps —

1. Segment and ispec attributes without defined initial values are initialized to their corresponding values from the input message.
2. Automatic edit of keys occurs – numeric fields are validated, separators and decimal points are removed. Any errors are returned to the application client.
3. The database record corresponding to the specified keys is retrieved
4. The retrieved record is made available (for ispecs with a user interface, this corresponds with its display to the application client).

Ispec Cycle

The ispec cycle is a subset of the segment cycle (as shown in Figure 1.5) and consists of the processing of an input message by a single ispec. It is controlled by each individual ispec. The ispec cycle can also be called independently of the segment cycle, such as with external automatic entry processing.

1.2 Introduction to Problem Domain

Agile Business Suite (ABSuite) is an application development framework using which developers can develop their applications efficiently and in less time, without having to write thousands of lines of code. The developers focus only on the model of the application and the ABSuite framework generates code from that model – including class hierarchies, tables, views and databases (with assertions, validations, aggregations, constraints, triggers and etc.), UI's, executable code or logic and etc. – it generates the full, ready to use application.

Once the application is deployed, the Runtime environment monitors all user interactions as transactions and maintains various log files – System logs, Audit logs, Tracker logs and etc. These log files contain various information like - which user is sending the request, at what time the request is generated and served, what information is received, whether there were any exceptions and etc. For example, audit logs contain information on how the application is used by the customers (For example – what attributes are sent by the user as input to perform an insert operation in the database). System logs contain information on how

the application interacts with the deployment environment (Windows/MCP). For example – whether the system failed for a transaction because of insufficient system resources (like a printer not being available) or whether the transaction failed due to an invalid input from the user. The first kind of error information will be found in the system logs whereas the second type of error messages will be found in the audit logs.

As an example, consider an IRCTC application developed using ABSuite. The IRCTC application will constantly interact with the runtime environment to handle client requests. Each client can be of different technology and can have hundreds of users to service. For example, we can have 200 users using a JSP client, 400 users using an ASP.Net client, 300 user using a WPF client and so on at a time. Under such loading conditions the runtime creates audit logs and system logs to keep track of all events (transactions and exceptions). A typical usage can lead to a 300 MB log file consisting of 10 million records Keeping this scenario in mind, the ultimate objective of our work is to design and develop an automated tool that provides in depth analysis for all systems or applications developed using ABSuite based on the contents of audit and system logs. The tool should be able to analyze the different log files created and discover information that can help improve system performance and reduce system downtime. It should be able to present statistical results that define how the system performs in terms of usage patterns, peak loading conditions, number, type and severity of exceptions generated and etc. Some of the information that can be extracted include the following —

1. **Traffic Behavior and Usage Patterns**
   Information pertaining to traffic and usage patterns can provide insight into parameters such as distribution of system load over a particular duration of time, peak loading hours of the system, most frequently accessed modules of the system, most and least performed transactions during a particular timespan and etc.

2. **Mean Time to Response (MTTR)**
   Mean Time to Response is defined as the average time taken by a transaction to service a request. Information pertaining to response times of the transactions can help us estimate the overall response time of the system. This parameter can also identify transactions with abnormally high latencies.

3. **Mean time to Failure (MTTF)**
   Mean Time to Failure is defined as the average time duration between the occurrences of two successive failures of the same transactions. Information pertaining to failure times can help us estimate the overall failure time of the system. This information can then be used to predict the time, type, severity and average service time of future failures.

4. **Error conditions and causes**

This gives information pertaining to different types of errors generated by the application and root causes of those errors. It can also indicate whether some new type of error has been encountered.

5. **System abnormalities and bugs**
   Hidden bugs are often introduced during the development cycle of a particular application. These bugs are rare and often result in unexpected system behavior. We hope to identify these bugs during our analysis.

Information gained from the log files can help us to improve system performance. The mined information can be used to statistically model the most frequent transactions occurring in a system. Statistical modeling includes the analysis of the transactions based on their probabilities of occurrences and probability distributions. High probabilities indicate heavily used modules or frequently occurring transactions. This will help to optimize the system from a transaction point of view. Other parameters that can be improved include the following —

1. **Object Pooling or Caching**
   Identification of the heavily shared objects in the system can help to better design the system so that performance is increased and response time is decreased.

2. **Allocation of Processing Power**
   Processing power is a limited resource when the system is under heavy loads. In such situations the decision to allocate processing power to which task or module becomes important

3. **Bandwidth Allocation**
   Identification of modules that are frequently used or accessed by majority of clients can help with the decision of whether to allocate more bandwidth to some modules than others. This will increase response time of the system.

4. **Scheduling of maintenance jobs**
   Maintenance and system downtime have to be reduced to a minimum so that system availability is high. It is preferable for the application to run most of the time without suffering from any performance degradation.

1.2.1 Issues and Challenges involved

Some of the major challenges encountered during the course of the project are discussed below —

1. It has been observed that the log files for an application generated for a single day can contain up to 10 million records each. Processing of such voluminous data falls under

the category of Data Mining and Big Data and requires its own set of algorithms. We have to ensure that System.OutOfMemory Exceptions and memory leaks are avoided when dealing with such large datasets. Data cleaning or preprocessing becomes important if we want to avoid bugs inherently present in the logs (such as presence of ‘ character that prevents proper parsing of a date-time object).

2. The literature and scientific community describe many different types of log files. Based on the content of the log files, we can have structured data (e.g. – data organized as a table which can be queried for information) or unstructured data (e.g. – news articles and email body). However, the log files from ABSuite Runtime are considered as semi-structured data, consisting of both a structured part and an unstructured part. Processing of such data is a problem that has to be carefully dealt with.

3. ABSuite Runtime generates many different types of log files, each with its own set of attributes. We have to carefully identify which log files are relevant for analysis, which set of attributes have to be considered and which set or records are relevant for the analysis.

4. Open sourced tools like Hadoop and Weka are suitable for structured data like web logs. We have to decide whether they work for our semi structured logs with minor modifications.

5. ABSuite Runtime generates various kinds of logs during the execution of an application. Each of the generated logs have unique formats different from each other. The format of each log file is specific and unique to Unisys corporation. This uniqueness in format introduces new challenges to the processing task.

6. Different stages of the analysis process give us different results. It is a big challenge to identify which combinations of those results will help us in adopting better design strategies for the application. It is also a big task to formulate and verify hypothesis inferred from different combinations of the results.

Chapter 2

Literature Survey

In the domain of computer science, Log Analysis is a scientific technique seeking to make sense out of computer-generated records (also called log or audit trail records). A log typically comprises of time sequenced stream of messages which are either stored on files or directed as a network stream to a log collector. Logs are the output of a process known as Data Logging. Data logging is a feature added to most systems and applications in order to check the runtime behavior of the system and to collect information about different events of the system. In the field of Databases and Transaction Management, data logging especially plays a crucial role in error recovery and data backup.

Logs are produced by various systems like - operating systems, database systems, networking devices, security systems, websites, e-commerce applications, banking applications, health care applications, and etc. These log files often contain large heaps of varied information. The log files of a particular system are a great source of knowledge if one wishes to analyze the runtime behavior of the system. However, the information contained in the logs is so huge that it is not possible to manually identify useful patterns and events. Standard techniques often fail when processing such large sources of information. Thus, the task of analyzing such vast amounts of data falls under the category of data mining. There are many articles and books available that describe data mining. [1] presents a comprehensive study of data mining from a database researcher's point of view. It discusses various issues related to data mining and various uses of data mining in different domains. [2] discusses various concepts and approaches used in data mining in detail. [3] discusses fundamental theory and goals of log file analysis. It also presents the current state of technology and practices in log file analysis and discusses various limitations and drawbacks of current log analysis products. Case studies relating to the analysis of Cisco NetFlow and HTTP server logs are also presented. The paper also proposes the requirements and design strategy of a universal log analyzer that uses data mining concepts to generate useful analysis.

The scientific community has a plethora of literature relating to log analysis. The most common among them relate to the analysis of web server logs which are produced by website servers. These logs are of various types and contain immense information like user preferences, usage patterns, search preferences and etc. There are many techniques to mine user profiles, preferences and behavioral aspects from historical data stored in Web

Logs. [4] discusses identification of user sessions from server access logs of a website to personalize web content according to different users. It uses a new clustering algorithm - Competitive Agglomeration for Relation Data - to group similar users based on different criteria. [5] presents similar work and discusses a framework for mining web user navigation patterns in order to develop personalization and recommender systems. [6] describes a dynamic approach to usage-based web personalization taking into account the full spectrum of Web mining techniques and activities. [7] introduces click stream data and proposes an effective and scalable technique for web personalization based on association rule mining from web usage data. [8] introduces the concept of frequent pattern mining from web logs to obtain information about the navigational behavior of the users. This paper introduces three pattern mining approaches based on page sets, page sequences and page graphs. [9] presents a survey of the use of Web mining for Web personalization. The paper also reviews some of the most common methods used, along with a brief overview of the most popular tools and applications available from software vendors. [10] introduces the problem of retrieval of irrelevant, redundant and inaccurate results when a user queries for a particular topic of interest on the World Wide Web. It defines and uses Web Mining to extract useful information and behavioral aspects of users using the website. [11] introduces search log data in relation to a web search. It discusses a way to mine major subtopics of a user query to a search engine so that more accurate search results are presented to the user. Specifically, the paper presents two concepts - ”one subtopic per search” and ”subtopic clarification by keyword” - and describes a novel clustering algorithm that uses these concepts to mine major subtopics of a user query. [12] presents an in-depth analysis of Web Logs of NASA website to find information like top errors, potential visitors and etc. which help the system administrator and Web designers to improve their system by determining occurred system errors, corrupted and broken links and etc. [13] demonstrates the capabilities of Correspondence Analysis (a novel data analysis method) on web log statistics for the examination of user behavior and preferences. Detection of user navigation paths is discussed in [14]. The paper explains the design and implementation of a profiler to capture client’s selected links. It also uses a novel clustering algorithm to cluster information gained by profiling different users. [15] presents a similar work for net traffic analysis, economical web site administration, website modifications, system improvement and personalization and business intelligence. [16] studies the problem of mining access patterns (similar to user navigation patterns) from Web logs efficiently. The paper discusses a novel data structure, called Web Access Pattern Tree to efficiently mine access patterns from web logs. Web Server Logs are usually noisy and hence require a preprocessing stage. After preprocessing the logs are ready for the actual analysis step. [17], [18], [19], [21] and [22] discuss various preprocessing and data preparation methods for web usage mining. [20] discusses several data preparation techniques in order to identify unique users and user sessions. The paper also presents a method to divide user sessions into semantically

meaningful transactions.

There are literatures that deal with log files other than Web Server Log Files like Transaction Logs, logs generated by a Security System, logs generated at a network device, logs generated by a library management system and etc. [23] presents a case study conducted at North Carolina State University. In this study, transaction logs of patrons searching an on-line catalog were analyzed to determine failure rates, usage patterns and causes of problems. This work is most closely related to our work in terms of the objective of log file analysis. [24] presents a related work to identify related journals through log analysis. [25] presents Event Logs obtained from various devices using the BSD sysLog protocol and discusses techniques of fault detection and anomaly detection using profiling.[26] presents error log analysis to demonstrate the presence of atleast two error processes in the logs. [27] discusses system logs generated by a system and categorization of different operating conditions for automatic system management. [28] discusses an interesting application of log file analysis to study the thought process of students playing educational games. Most of the above works require that analysts know what they are looking for in the logs beforehand, however this is not the case always. [29] presents an example of a security system where the analyst does not have prior domain specific knowledge and thus does not know what to look for. The authors present a method to mine interesting patterns in such a situation.

As part of our literature survey, we have gone through some of the commonly available log analysis products, including open sourced, proprietary and products hosted as a service. Most of the products like AWStats, Open Web Analytics, Piwik, Webalizer, Mint, Sawmill, Splunk, Urchin, Adobe Analytics, Google Analytics and etc. perform comprehensive analysis of only web server logs like access logs, error logs and etc. The advantages of these products is that they can be easily configured for any type of server logs like W3C's Common Log Format, Apache Server Logs (XLF or ELF), IIS Log formats and etc. These products differ in the technology used to perform the analysis. For example some of the products use Cookies and Javascript to track user interaction while others are web logs based and still others use a combination of PHP and Page Tagging. The results obtained are used to optimize web based applications, perform Web Usage Mining and deduce valuable information related to how the website is being used by different users. Some of the products like AWStats provide support for custom logs, however for such situations one has to provide the schema definition of each log file in a language (like Pearl) that is supported by the product. Using these type of products thus has three issues - First is the prior knowledge of the log files, Second is the product specific knowledge and operation conditions and third is the heterogeneous nature of log files. Detailed comparison of different products is provided in the table 2.1 below —

Table 2.1: Product Survey

PRODUCT	DEVELOPER	ANALYSIS METHOD	PLATFORM	WEBSITE
AWStats	Open Sourced	Web Logs based	Perl	awstats.org
Open Web Analytics	Open Sourced	Javascript or PHP Page Tag	PHP	-
PiWik	Open Sourced	Javascript or PHP Page Tag or Web Logs based	PHP	-
Webalizer	Open Sourced	Web Logs based	C	-
Mint	Mint	Cookies via Javascript	PHP	-
Sawmill	FlowerFire Inc.	Cookie via Javascript or Web Logs based	Windows or Linux	-
Splunk	Splunk Inc.	Web Logs based	Windows or Linux	splunk.com
Urchin	Google	Cookies and Logs	Windows or Linux	google.com
Adobe Analytics	Adobe Systems	Cookies via Javascript	SaaS	adobe.com
Google Analytics	Google	Cookies via Javascript	SaaS	google.com

Chapter 3

Proposed Algorithms

3.1 General Methodology

The ABSuite Runtime treats all user interaction with the application as transactions. For example, if a user wants to check his bank account balance then he will have to submit a query using a front-end client like Windows Forms, ASP.Net and etc. This query will be treated by the Runtime as a transaction that gets executed against a runtime database. Internally each transaction is represented by a unique Ispec (Interface Specification). The logs generated by the ABSuite Runtime capture all transactions in the form of their respective Ispecs. For our analysis, we have used two kinds of log files —

1. **Audit Logs**

   These files maintain information on how the application is used by the customers. For example – what attributes are sent by the user as input to perform an insert operation in the database. These files are responsible for keeping track of all ABSuite elements that interact with the Runtime Application DB including – Ispecs, Reports, HUB transactions and etc. Typical usage of an application results in audit logs that are approximately 500 MB in size and consisting of 10 million records. The structure of an audit log is shown in Figure 3.1

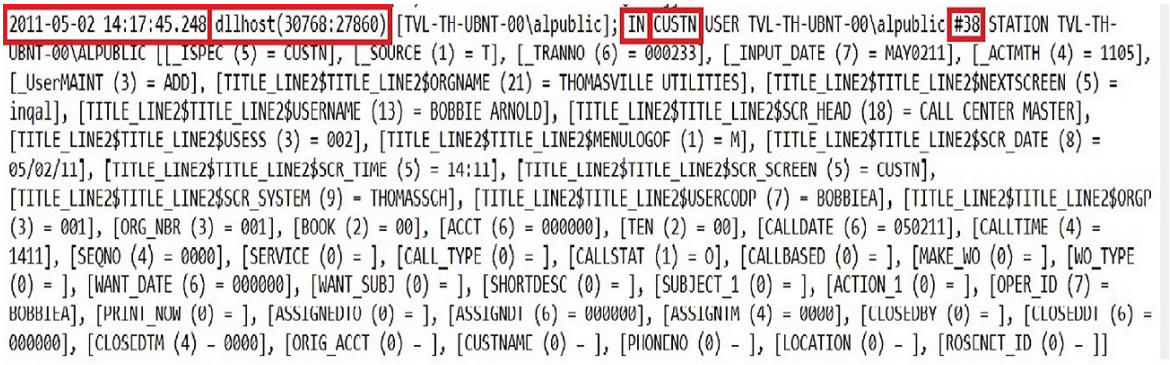


Figure 3.1: A record present in the Audit log of an ABSuite application

Following are the major attributes of the audit files used in our analysis —

A) Timestamp

The timestamp represents the date and time at which the transaction occurred. It follows the yyyy-mm-dd hh:mm: ss.fff format. This format is application specific.

B) Process ID : Thread ID

Every transaction executes as a process hosted by a DLL. This combination of process and thread Ids represents the thread and the containing process Id responsible for executing the transaction.

C) Mode of Transaction

This field represents whether a transaction performs a request operation to (IN Mode) or a reply operation (OUT Mode) from the Runtime DB. This field can also be used to determine whether a transaction is caused by an Ispec or due to any other element like a report.

D) Element Name

This field represents the name of the element performing the transaction. For Ispecs and reports, this value represents the unique name of the element. For HUB transactions, this value is set to ”HUB”.

E) Session ID

This value represents the unique session Id of the session during which the transaction is performed. In one particular session, we can have many different transactions being performed. But since each transaction is atomic, we will have the entire request-reply cycle in one session only i.e. It is not possible to have an Ispec with its IN mode in one session and OUT mode in a different session.

F) Other fields present in the Figure include login name of the user, station name of the user, IP address of the terminal used (if present), body of the transaction performed and etc.

2. **System Logs**

System logs maintain information on how the application interacts with the deployment environment. For example – whether the system failed for a transaction because of insufficient system resources (like a printer not being available) or whether the transaction failed due to invalid input from the user. The first kind of error information will be found in the system logs whereas the second type of error messages will be found in the audit logs. The structure of a system log is shown in Figure 3.2

Some of the important attributes of the System logs are discussed below —

A) Timestamp

```
2015-04-27 19:17:06.334 SAS01AUT04(9704:7996) [CONSORCIOSALUD\appuser]; ReportOutputStream uses codepage 1252
2015-04-27 19:17:07.014 SAS01AUT04(9704:7996) [CONSORCIOSALUD\appuser]; OutputStream Default OutputStream running to print after
RELEASE.
2015-04-27 19:17:07.015 SAS01AUT04(9704:7996) [CONSORCIOSALUD\appuser]; Nothing for _PrintRoutine() to print from Default
OutputStream
2015-04-27 19:17:13.223 AFILDBCS(29788:13964) [CONSORCIOSALUD\AppUser]; The following exception occurred at "SQLDbHelper.ExecuteCmd"
=====================================
Exception:    System.Data.SqlClient.SqlException
Message:      Transaction (Process ID 108) was deadlocked on lock resources with another process and has been chosen as the deadlock
victim. Rerun the transaction.
Source:       .Net SqlClient Data Provider
TargetSite:   Void OnError(System.Data.SqlClient.SqlException, Boolean)
StackTrace:
   at System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection)
```

Figure 3.2: A record present in the System log of an ABSuite application

The timestamp represents the date and time at which the event (whether normal processing or generation of exception condition) occurred. It follows the yyyy-mm-dd hh:mm: ss.fff format. This format is application specific.

B) Process ID: Thread ID

Every transaction executes as a process hosted by a DLL. This combination of process and thread Ids represents the thread and the containing process Id responsible for executing the transaction (whether normal processing or generation of exception condition).

C) For transactions that completed successfully, the System logs maintain information on how the application interacted with the deployment environment (or the underlying operating system). For transactions resulting in exceptions, the System logs also contain information on the exception like – exception name and message, corresponding stack trace and etc as shown in Figure 3.2

The ABSuite Log Analyzer is a tool that extracts transaction information by analyzing the Ispec information present in the generated logs in three steps. We briefly discuss the general methodology for each analysis step below.

The first step in the analysis process is the gathering of basic Ispec information such as number of different Ispecs present in the system, the number of times each Ispec triggered during the span of the log files, probability distribution of each Ispec and etc. This information helps in identifying system load and peak loading conditions, system usage patterns and most commonly occurring transactions. This step uses the Audit logs only.

Mean Time to respond (MTTR) is an important parameter to measure the performance of any system. The second stage of our analysis is Response Time Analysis which plays a crucial role in evaluating the responsiveness and availability of the systems developed using ABSuite. MTTR is defined as the time duration between sending the request to the runtime server and receiving the corresponding reply from the runtime server. For our applications, Ispecs are responsible for performing all transactions. These transactions occur in the form of request-reply pairs in the audit logs of the system. For Example – if a client wants to

see the number of products available in his inventory then this query is fired as an Ispec sending a request to the server (IN mode). The reply from the server is also obtained from the same Ispec (OUT mode) thus constituting a transaction. We will use the audit logs for this stage of analysis since they contain the required information like when a request was made by an Ispec, which process was hosting the Ispec, which client triggered the request, what information was passed to the server, when did the server reply back, what information was transmitted by the server and etc (Refer Figure 3.1).

One of our key motivations is to track Ispecs or transactions that have caused exceptions in the system. These exceptions can occur due to reasons like improper implementation of the Ispec, unavailability of system resources, missing DLLs and references and etc. The third stage deals with mapping exceptions to their corresponding Ispecs which will help us to identify "faulty" Ispecs that result in abnormal behavior of the overall system. In addition, we can also get information like - how many exceptions were caused by a transaction, what was the type and severity of each exception, how many exceptions were caused due to lack of system resources, Is there a need to increase system resources and etc. Based on the above information we can make high level decisions like can we improve system design to minimize exceptions, how many additional resources of each type are needed to minimize exceptions due to unavailability of system resources and etc. For this stage of analysis, we require both System logs and Audit logs. The System logs are responsible for tracking any exceptions that the system throws during runtime (Refer Figure 3.2). For example – for a particular transaction request if it is found that the application server is off-line then Runtime generates an exception which gets logged in the System logs. Both System and Audit logs have a time stamp value and the Id of the host process which can be used to map exceptions (in Systems logs) to the potential Ispec (in the audit log) that caused it.

3.2 Proposed Algorithm for Basic Ispec Analysis

The algorithm used at this stage of analysis is shown in Algorithm 1 below —

The input to the algorithm is the complete path of the directory which contains the required audit logs. The algorithm starts with a preprocessing method which addresses two issues —

1. Cleaning each record of the audit files and handling improper parsing of time stamp values due to the presence of extra characters.

2. Obtaining a list of distinct Ispecs present in the Audit logs.

For each Ispec, the algorithm finds out how many records of the Audit logs belong to that Ispec. The output of this algorithm is a list of Ispecs with their associated count values. The count values represent how many times a particular Ispec was triggered during the span

Algorithm 1 Algorithm for Basic Ispec Analysis

Input: Audit log directory path.
Output: Trigger Count for each Ispec.

```
Initialize IspecList during preprocessing
for all ISPEC in IspecList do
  triggerCount ← 0
  for all AUDIT_FILE in AUDIT_LOGS do
    for all RECORDS in AUDIT_FILE do
      if ((RECORD.MODE = "IN" OR RECORD.MODE = "OUT") AND
      (RECORD.NAME = ISPEC)) then
        triggerCount ← triggerCount + 1
      end if
    end for
  end for
end for
```

of the Audit logs. They also represent how many times the transaction related to the Ispec was performed by the users of the application. Hence, we obtain the usage pattern of each transaction present in the audit logs. The next sub-sections presents some more parameters derived from the usage patterns like transaction frequency and distribution.

3.3 Proposed Algorithm for Response Time Analysis

The algorithm used at this stage of analysis is shown in Algorithm 2 below —

The input to the algorithm is the complete path of the directory which contains the required audit logs. The algorithm starts with a preprocessing method which addresses three issues —

1. Cleaning each record of the audit files and handling improper parsing of time stamp values due to the presence of extra characters.

2. Obtaining a list of distinct Ispecs present in the Audit logs.

3. For each Ispec, create lists of processes that host the Ispec. For example - If an Ispec is hosted by n processes then this step creates n lists belonging to each process. These list are used to store Ispecs with IN mode (transaction requests).

The algorithm proceeds by segregating transaction details for each Ispec according to the processes hosting the transactions. For example – let there be an Ispec I which is hosted by different processes say – $P_1, P_2..., P_n$. Create n lists for the n processes of I and then calculate response times for each process from the n lists. For Example - let process P_i have

x transactions with response times – $t_1, t_2, ..., t_x$. Then

$$MTTR(P_i) = \frac{1}{x} * \sum_{i=1}^{x} t_i$$

After finding MTTR values for each of the n processes of the Ispec I, we can find the MTTR value of the Ispec I as

$$MTTR(I) = \frac{1}{n} * \sum_{i=1}^{n} MTTR(P_i)$$

The benefit of this approach is that it enables us to find MTTR values in terms of the processes hosting the Ispec transactions thus enabling better (process level) granularity. This helps us to find out which processes contribute more to the MTTR values of the Ispec thus identifying those processes as "abnormal".

Algorithm 2 Algorithm for Response Time Analysis

Input: Audit log directory path.
Output: List of response times for each contributing process of the Ispec.

```
Initialize IspecList during preprocessing
Initialize ProcessList for all Ispecs during preprocessing
for all ISPEC in IspecList do
  for all AUDIT_FILE in AUDIT_LOGS do
    for all RECORDS in AUDIT_FILE do
      if ((RECORD.MODE = "IN") AND (RECORD.NAME = ISPEC)) then
        Add RECORD in ProcessList where ProcessList =
        RECORD.PROCESS_ID
      end if
      if ((RECORD.MODE = "OUT") AND (RECORD.NAME = ISPEC))
      then
        Get nearest IN_RECORD in ProcessList where ProcessList =
        RECORD.PROCESS_ID
        Calculate ResponseTime ← RECORD.TIMESTAMP −
        IN_RECORD.TIMESTAMP
        Remove IN_RECORD from ProcessList where ProcessList =
        RECORD.PROCESS_ID
      end if
    end for
  end for
end for
```

3.4 Proposed Algorithm for Exception to Ispec Mapping

The algorithm used at this stage of analysis is shown in Algorithm 3 below —

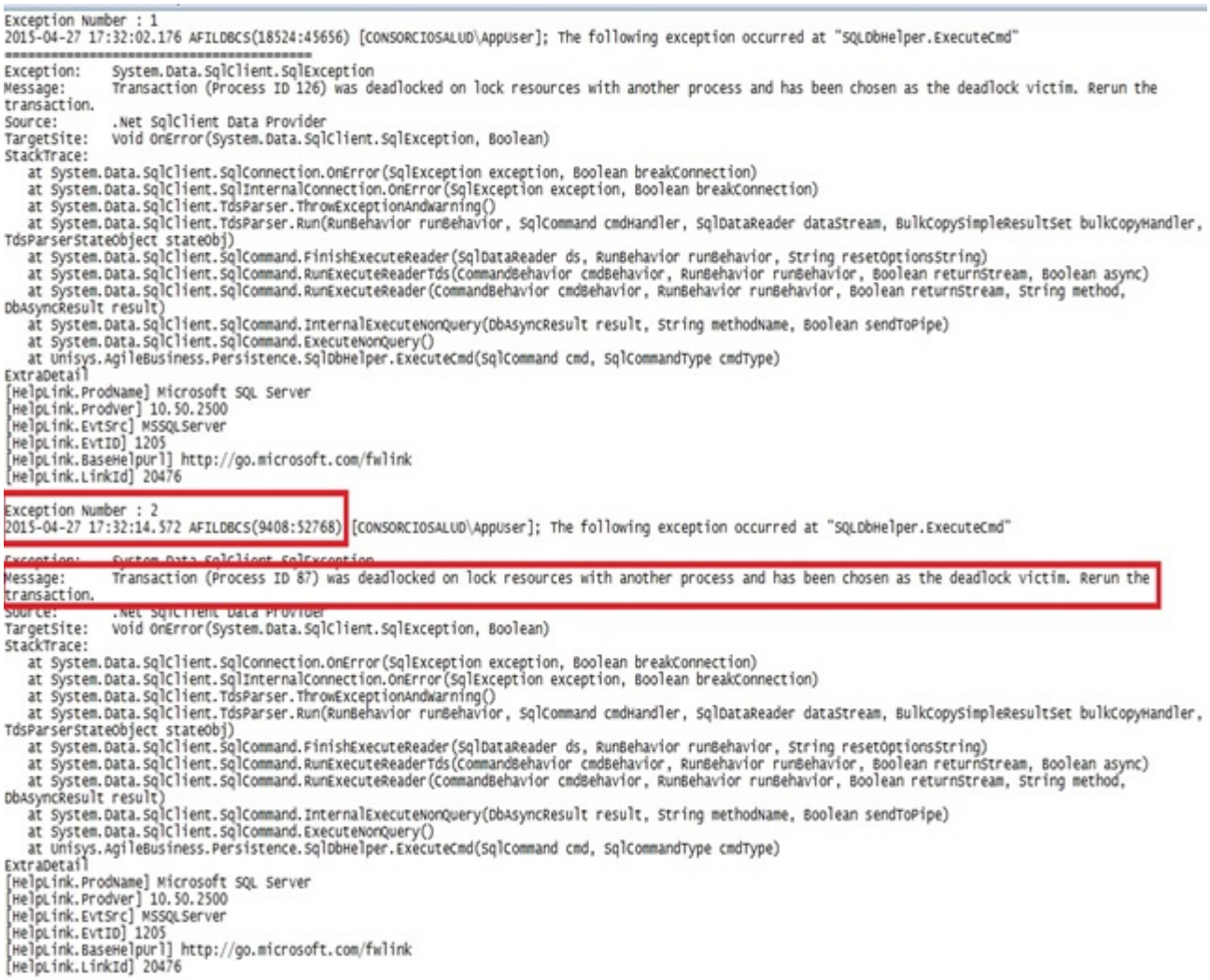
Algorithm 3 Algorithm for Exception to Ispec Mapping

Input: Audit log directory path and System log directory path.
Output: List of exceptions tagged with their corresponding elements.

```
Create ExceptionsList during preprocessing of System Logs
Initialize IspecList during preprocessing of Audit Logs
for all EXCEPTION in ExceptionsList do
  exceptionTimestamp ← EXCEPTION.TIMESTAMP
  processId ← EXCEPTION.PROCESS_ID
  for all AUDIT_FILE in AUDIT_LOGS do
    for all RECORDS in AUDIT_FILE do
      Get Record closest to exceptionTimestamp where
      RECORD.PROCESS_ID = processId
      if (RECORD.MODE = "IN" OR RECORD.MODE = "OUT") then
        Tag RECORD as Exception caused due to ispec
      end if
      Tag RECORD as Exception caused due to other elements
    end for
  end for
end for
```

The algorithm at this stage of analysis first uses a preprocessing step to read the System logs of a deployed system and extract all exceptions into a temporary file. A snapshot of this temporary file is shown in Figure 3.3 below —.

This temporary file serves as a dictionary and contains information on various exceptions that were generated. Another use of this file is to provide time stamp and process Ids needed for mapping the exception to the corresponding Ispec or non - Ispec elements. The exceptions are then mapped to their corresponding elements in the audit logs.



```
Exception Number : 1
2015-04-27 17:32:02.176 AFILDBCS(18524:45656) [CONSORCIOSALUD\AppUser]; The following exception occurred at "SQLDbHelper.ExecuteCmd"
=======================================
Exception:    System.Data.SqlClient.SqlException
Message:      Transaction (Process ID 126) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the
transaction.
Source:       .Net SqlClient Data Provider
TargetSite:   Void OnError(System.Data.SqlClient.SqlException, Boolean)
StackTrace:
   at System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection)
   at System.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean breakConnection)
   at System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning()
   at System.Data.SqlClient.TdsParser.Run(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopySimpleResultSet bulkCopyHandler,
TdsParserStateObject stateObj)
   at System.Data.SqlClient.SqlCommand.FinishExecuteReader(SqlDataReader ds, RunBehavior runBehavior, String resetOptionsString)
   at System.Data.SqlClient.SqlCommand.RunExecuteReaderTds(CommandBehavior cmdBehavior, RunBehavior runBehavior, Boolean returnStream, Boolean async)
   at System.Data.SqlClient.SqlCommand.RunExecuteReader(CommandBehavior cmdBehavior, RunBehavior runBehavior, Boolean returnStream, String method,
DbAsyncResult result)
   at System.Data.SqlClient.SqlCommand.InternalExecuteNonQuery(DbAsyncResult result, String methodName, Boolean sendToPipe)
   at System.Data.SqlClient.SqlCommand.ExecuteNonQuery()
   at Unisys.AgileBusiness.Persistence.SqlDbHelper.ExecuteCmd(SqlCommand cmd, SqlCommandType cmdType)
ExtraDetail
[HelpLink.ProdName] Microsoft SQL Server
[HelpLink.ProdVer] 10.50.2500
[HelpLink.EvtSrc] MSSQLServer
[HelpLink.EvtID] 1205
[HelpLink.BaseHelpUrl] http://go.microsoft.com/fwlink
[HelpLink.LinkId] 20476

Exception Number : 2
2015-04-27 17:32:14.572 AFILDBCS(9408:52768) [CONSORCIOSALUD\AppUser]; The following exception occurred at "SQLDbHelper.ExecuteCmd"
=======================================
Exception:    System.Data.SqlClient.SqlException
Message:      Transaction (Process ID 87) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the
transaction.
Source:       .Net SqlClient Data Provider
TargetSite:   Void OnError(System.Data.SqlClient.SqlException, Boolean)
StackTrace:
   at System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection)
   at System.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean breakConnection)
   at System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning()
   at System.Data.SqlClient.TdsParser.Run(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopySimpleResultSet bulkCopyHandler,
TdsParserStateObject stateObj)
   at System.Data.SqlClient.SqlCommand.FinishExecuteReader(SqlDataReader ds, RunBehavior runBehavior, String resetOptionsString)
   at System.Data.SqlClient.SqlCommand.RunExecuteReaderTds(CommandBehavior cmdBehavior, RunBehavior runBehavior, Boolean returnStream, Boolean async)
   at System.Data.SqlClient.SqlCommand.RunExecuteReader(CommandBehavior cmdBehavior, RunBehavior runBehavior, Boolean returnStream, String method,
DbAsyncResult result)
   at System.Data.SqlClient.SqlCommand.InternalExecuteNonQuery(DbAsyncResult result, String methodName, Boolean sendToPipe)
   at System.Data.SqlClient.SqlCommand.ExecuteNonQuery()
   at Unisys.AgileBusiness.Persistence.SqlDbHelper.ExecuteCmd(SqlCommand cmd, SqlCommandType cmdType)
ExtraDetail
[HelpLink.ProdName] Microsoft SQL Server
[HelpLink.ProdVer] 10.50.2500
[HelpLink.EvtSrc] MSSQLServer
[HelpLink.EvtID] 1205
[HelpLink.BaseHelpUrl] http://go.microsoft.com/fwlink
[HelpLink.LinkId] 20476
```

Figure 3.3: A snapshot of the temporary file showing exception details for a system.

3.5 Implementation

We have used two sets of log files, each obtained from two different applications. One of the applications represents a system that is heavily used during typical working hours (8 am to 9 pm) while the other represents a system that is frequently used only for a small duration (2 hours). We have used both sets of log files to compare the results of our ABSuite Log Analyzer tool for two contrasting systems.

The ABSuite Log Analyzer is intended to run as a windows form application on windows platform. Thus, C# was chosen as the programming language to perform analysis as it has extensive support for string manipulation, date time object conversions and compatibility with .Net Framework which is required to generate graphs. Microsoft Visual Studio was used as the development environment because of its support for .Net framework and windows form applications.

Chapter 4
Experimental Results and Observations

4.1 Experimental Results and Observations for Basic Ispec Mapping

Basic Ispec Analysis deals with finding statistical parameters like trigger count, percentage, trigger frequency and average ispecs triggered per day for each transaction. These values are further used to estimate the distribution of each Ispec. Each of the above parameters are defined as below —

1. Trigger Count

   TriggerCount represents the number of times an Ispec triggered during the given log span.

2. Percentage

   $Percentage = TriggerCount_i / TotalIspecCount$

   where *TriggerCount* represents the number of times the Ispec triggered and *TotalIspecCount* represents sum of *TriggerCount* for all Ispecs.

3. Trigger Frequency

   $Trigger frequency = TriggerCount_i / AuditLogSpan$

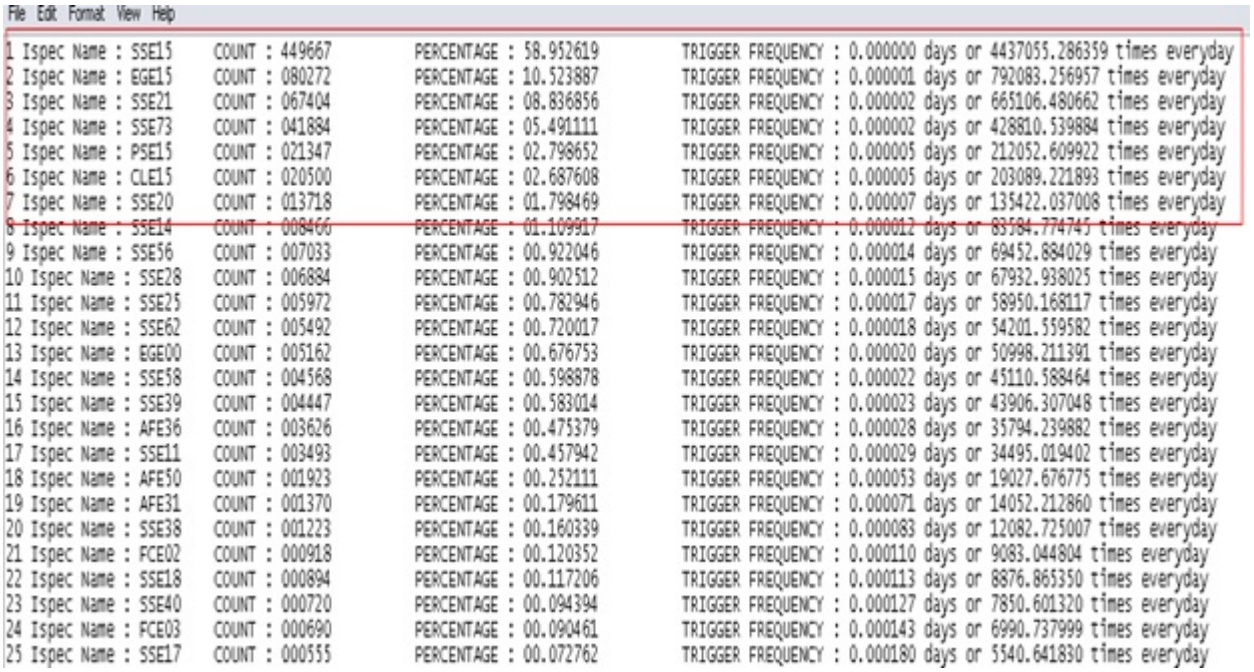
   where *AuditLogSpan* represents the duration for which the audit logs have been created.

   Trigger frequency for an Ispec measures how often a particular transaction or Ispec is triggered. It is a measure of heavily used modules of the system.

4. Average number of Ispecs triggered per day

   $AvgIspecs = TotalIspecCount / AuditLogSpan$

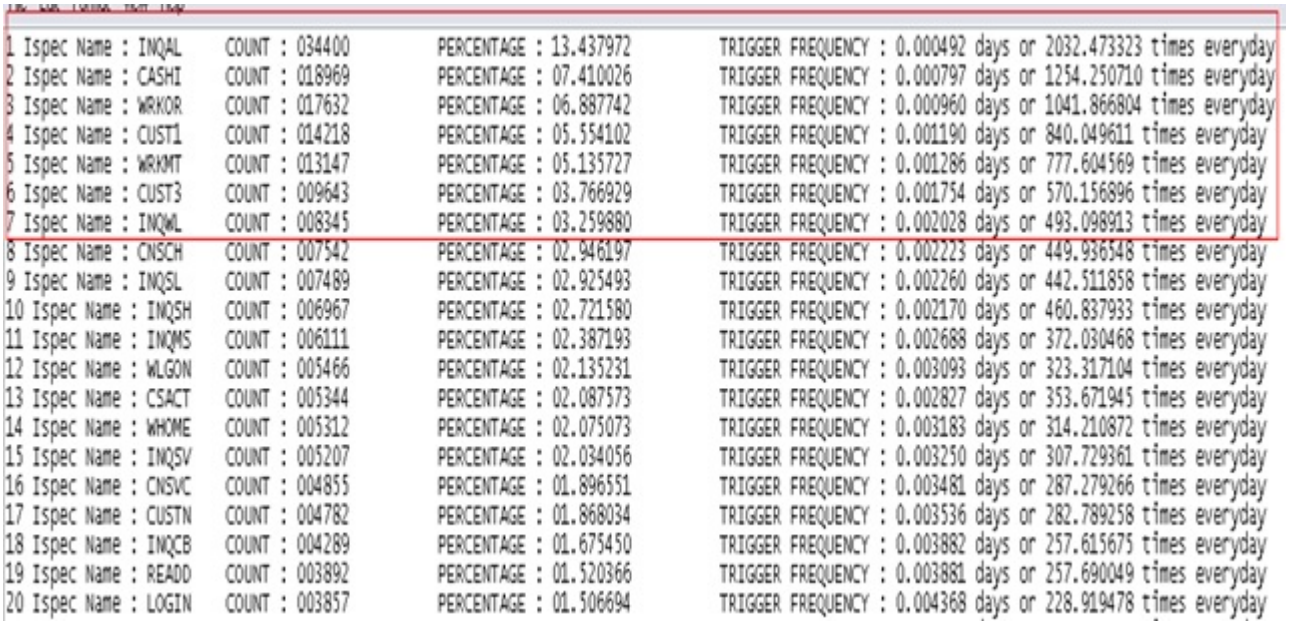
   where *AvgIspecs* quantifies the usage pattern on a daily basis. This parameter measures the system load for a given day.



File Edit Format View Help

```
1 Ispec Name : SSE15    COUNT : 449667    PERCENTAGE : 58.952619    TRIGGER FREQUENCY : 0.000000 days or 4437055.286359 times everyday
2 Ispec Name : EGE15    COUNT : 080272    PERCENTAGE : 10.523887    TRIGGER FREQUENCY : 0.000001 days or 792083.256957 times everyday
3 Ispec Name : SSE21    COUNT : 067404    PERCENTAGE : 08.836856    TRIGGER FREQUENCY : 0.000002 days or 665106.480662 times everyday
4 Ispec Name : SSE73    COUNT : 041884    PERCENTAGE : 05.491111    TRIGGER FREQUENCY : 0.000002 days or 428810.539884 times everyday
5 Ispec Name : PSE15    COUNT : 021347    PERCENTAGE : 02.798652    TRIGGER FREQUENCY : 0.000005 days or 212052.609922 times everyday
6 Ispec Name : CLE15    COUNT : 020500    PERCENTAGE : 02.687608    TRIGGER FREQUENCY : 0.000005 days or 203089.221893 times everyday
7 Ispec Name : SSE20    COUNT : 013718    PERCENTAGE : 01.798469    TRIGGER FREQUENCY : 0.000007 days or 135422.037008 times everyday
8 Ispec Name : SSE14    COUNT : 008466    PERCENTAGE : 01.109917    TRIGGER FREQUENCY : 0.000012 days or 83584.774745 times everyday
9 Ispec Name : SSE56    COUNT : 007033    PERCENTAGE : 00.922046    TRIGGER FREQUENCY : 0.000014 days or 69452.884029 times everyday
10 Ispec Name : SSE28   COUNT : 006884    PERCENTAGE : 00.902512    TRIGGER FREQUENCY : 0.000015 days or 67932.938025 times everyday
11 Ispec Name : SSE25   COUNT : 005972    PERCENTAGE : 00.782946    TRIGGER FREQUENCY : 0.000017 days or 58950.168117 times everyday
12 Ispec Name : SSE62   COUNT : 005492    PERCENTAGE : 00.720017    TRIGGER FREQUENCY : 0.000018 days or 54201.559582 times everyday
13 Ispec Name : EGE00   COUNT : 005162    PERCENTAGE : 00.676753    TRIGGER FREQUENCY : 0.000020 days or 50998.211391 times everyday
14 Ispec Name : SSE58   COUNT : 004568    PERCENTAGE : 00.598878    TRIGGER FREQUENCY : 0.000022 days or 45110.588464 times everyday
15 Ispec Name : SSE39   COUNT : 004447    PERCENTAGE : 00.583014    TRIGGER FREQUENCY : 0.000023 days or 43906.307048 times everyday
16 Ispec Name : AFE36   COUNT : 003626    PERCENTAGE : 00.475379    TRIGGER FREQUENCY : 0.000028 days or 35794.239882 times everyday
17 Ispec Name : SSE11   COUNT : 003493    PERCENTAGE : 00.457942    TRIGGER FREQUENCY : 0.000029 days or 34495.019402 times everyday
18 Ispec Name : AFE30   COUNT : 001923    PERCENTAGE : 00.252111    TRIGGER FREQUENCY : 0.000053 days or 19027.676775 times everyday
19 Ispec Name : AFE31   COUNT : 001370    PERCENTAGE : 00.179611    TRIGGER FREQUENCY : 0.000071 days or 14052.212860 times everyday
20 Ispec Name : SSE38   COUNT : 001223    PERCENTAGE : 00.160339    TRIGGER FREQUENCY : 0.000083 days or 12082.725007 times everyday
21 Ispec Name : FCE02   COUNT : 000918    PERCENTAGE : 00.120352    TRIGGER FREQUENCY : 0.000110 days or 9083.044804 times everyday
22 Ispec Name : SSE18   COUNT : 000894    PERCENTAGE : 00.117206    TRIGGER FREQUENCY : 0.000113 days or 8876.865350 times everyday
23 Ispec Name : SSE40   COUNT : 000720    PERCENTAGE : 00.094394    TRIGGER FREQUENCY : 0.000127 days or 7850.601320 times everyday
24 Ispec Name : FCE03   COUNT : 000690    PERCENTAGE : 00.090461    TRIGGER FREQUENCY : 0.000143 days or 6990.737999 times everyday
25 Ispec Name : SSE17   COUNT : 000555    PERCENTAGE : 00.072762    TRIGGER FREQUENCY : 0.000180 days or 5540.641830 times everyday
```

Figure 4.1: Ispec details for Application 1



```
1 Ispec Name : INQAL    COUNT : 034400    PERCENTAGE : 13.437972    TRIGGER FREQUENCY : 0.000492 days or 2032.473323 times everyday
2 Ispec Name : CASHI    COUNT : 018969    PERCENTAGE : 07.410026    TRIGGER FREQUENCY : 0.000797 days or 1254.250710 times everyday
3 Ispec Name : WRKOR    COUNT : 017632    PERCENTAGE : 06.887742    TRIGGER FREQUENCY : 0.000960 days or 1041.866804 times everyday
4 Ispec Name : CUST1    COUNT : 014218    PERCENTAGE : 05.554102    TRIGGER FREQUENCY : 0.001190 days or 840.049611 times everyday
5 Ispec Name : WRKMT    COUNT : 013147    PERCENTAGE : 05.135727    TRIGGER FREQUENCY : 0.001286 days or 777.604569 times everyday
6 Ispec Name : CUST3    COUNT : 009643    PERCENTAGE : 03.766929    TRIGGER FREQUENCY : 0.001754 days or 570.156896 times everyday
7 Ispec Name : INQWL    COUNT : 008345    PERCENTAGE : 03.259880    TRIGGER FREQUENCY : 0.002028 days or 493.098913 times everyday
8 Ispec Name : CNSCH    COUNT : 007542    PERCENTAGE : 02.946197    TRIGGER FREQUENCY : 0.002223 days or 449.936548 times everyday
9 Ispec Name : INQSL    COUNT : 007489    PERCENTAGE : 02.925493    TRIGGER FREQUENCY : 0.002260 days or 442.511858 times everyday
10 Ispec Name : INQSH   COUNT : 006967    PERCENTAGE : 02.721580    TRIGGER FREQUENCY : 0.002170 days or 460.837933 times everyday
11 Ispec Name : INQMS   COUNT : 006111    PERCENTAGE : 02.387193    TRIGGER FREQUENCY : 0.002688 days or 372.030468 times everyday
12 Ispec Name : WLGON   COUNT : 005466    PERCENTAGE : 02.135231    TRIGGER FREQUENCY : 0.003093 days or 323.317104 times everyday
13 Ispec Name : CSACT   COUNT : 005344    PERCENTAGE : 02.087573    TRIGGER FREQUENCY : 0.002827 days or 353.671945 times everyday
14 Ispec Name : WHOME   COUNT : 005312    PERCENTAGE : 02.075073    TRIGGER FREQUENCY : 0.003183 days or 314.210872 times everyday
15 Ispec Name : INQSV   COUNT : 005207    PERCENTAGE : 02.034056    TRIGGER FREQUENCY : 0.003250 days or 307.729361 times everyday
16 Ispec Name : CNSVC   COUNT : 004855    PERCENTAGE : 01.896551    TRIGGER FREQUENCY : 0.003481 days or 287.279266 times everyday
17 Ispec Name : CUSTN   COUNT : 004782    PERCENTAGE : 01.868034    TRIGGER FREQUENCY : 0.003536 days or 282.789258 times everyday
18 Ispec Name : INQCB   COUNT : 004289    PERCENTAGE : 01.675450    TRIGGER FREQUENCY : 0.003882 days or 257.615675 times everyday
19 Ispec Name : READD   COUNT : 003892    PERCENTAGE : 01.520366    TRIGGER FREQUENCY : 0.003881 days or 257.690049 times everyday
20 Ispec Name : LOGIN   COUNT : 003857    PERCENTAGE : 01.506694    TRIGGER FREQUENCY : 0.004368 days or 228.919478 times everyday
```

Figure 4.2: Ispec details for Application 2

The results of this processing stage is shown in Figure 4.1 for Application 1 and Figure 4.2 for Application 2 above —

A table comparing the results of this analysis stage for the two applications is shown below in Table 4.1 —

Table 4.1: Basic Ispec Analysis for System 1 vs. System 2.

PARAMETERS	SYSTEM 1	SYSTEM 2
Audit log span (in days)	1	16
Number of files	275	55
Size of audit logs	2.62 GB	537 MB
Total Ispec activity	762760	255991
Number of distinct Ispecs	58	215
Avg. no. of Ispecs triggered per day	762760	15999.44
Number of frequent Ispecs	7 (out of 58: 12 %)	39 (out of 215: 18 %)
Most frequent Ispecs	SSE15, EGE15, SSE21	INQAL, CASHI, WRKOR
Trigger Counts for frequent Ispecs	449667, 80272, 67404	34400, 18969, 17632
Percentage	58.9, 10.5, 8.8	13.4, 7.4, 6.8
Trigger frequency (times per day)	449667, 80272, 67404	2032.4, 1254.2, 1041.8

4.1.1 Observations for Basic Ispec Analysis

Our initial tests seem to suggest that both systems are heavily used with system 1 being more heavily used. System 1 has only 7 frequent Ispecs whereas system 2 has 39 frequent Ispecs (Frequent ispecs are those Ispecs that constitute 90 % of the total Ispec activity of the system). In both the systems only a small percentage of the Ispecs contribute to the overall load (about 19 %). This suggests that user activity is concentrated only in these Ispec activities. In order to better visualize the above statistics, we provide the below graphs. Figure 4.3 and Figure 4.4 show the trigger count for each frequent Ispec in system 1 and system 2 respectively.

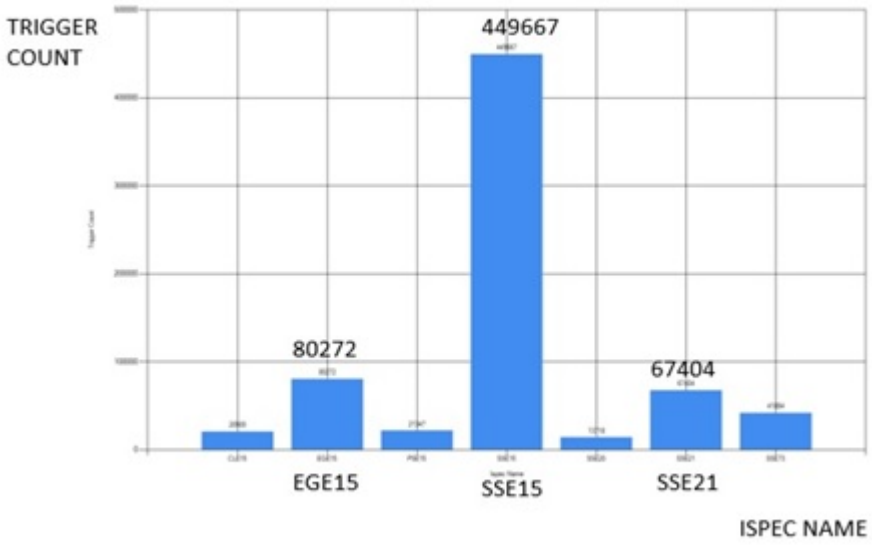


Figure 4.3: Frequent Ipecs for Application 1

To determine the distribution of Ispecs (or transactions) over time (24 hours) we present the graphs below. Figures 4.5, 4.6 and 4.7 show the Ispec distribution for the three most frequent Ispecs in system 1 whereas Figure 4.8, 4.9 and 4.10 show the same for system

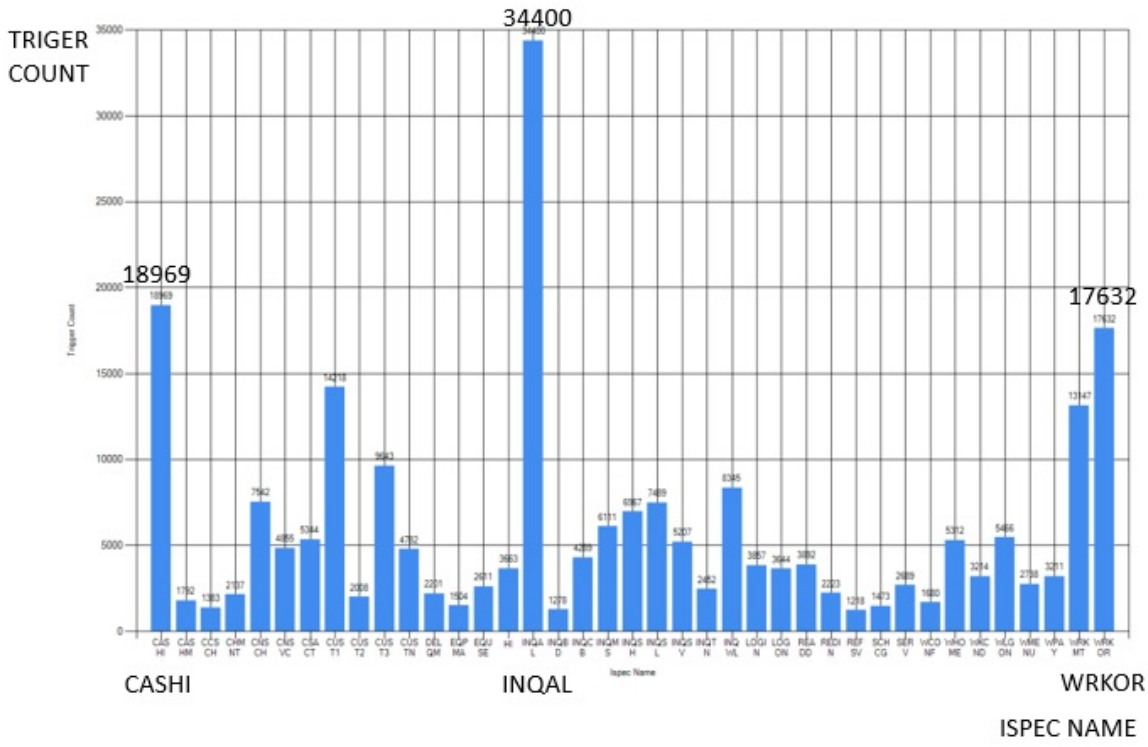


Figure 4.4: Frequent Ipecs for Application 2

2. The x-axis shows the 24-hour time line whereas y-axis shows the trigger count. These distributions were obtained for all frequent Ispecs and are primarily used to predict the probability of occurrence of a particular transaction during a given time interval.

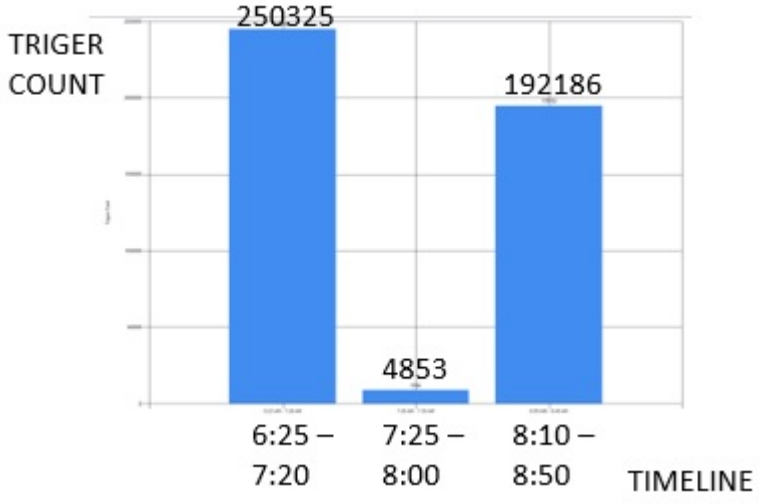


Figure 4.5: Ispec Distribution for SSE15 (System 1)

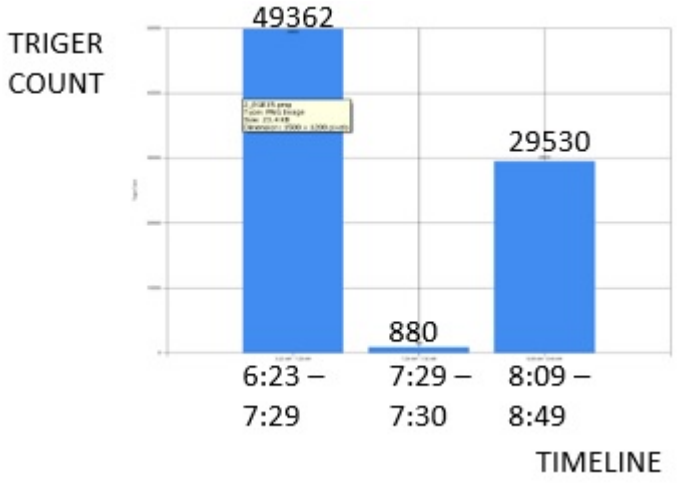


Figure 4.6: Ispec Distribution for EGE15 (System 1)

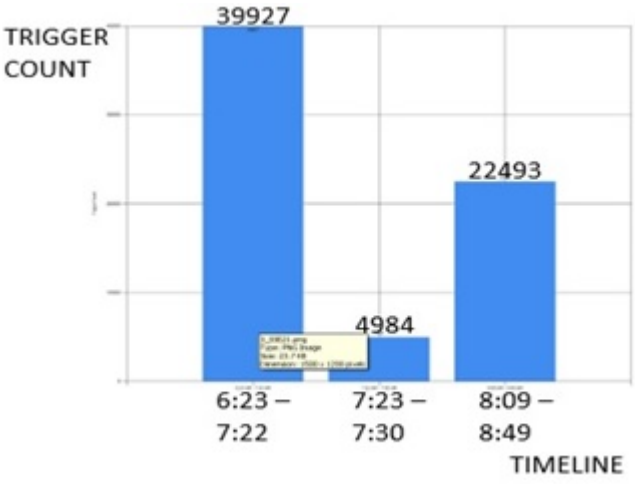


Figure 4.7: Ispec Distribution for SSE21 (System 1)

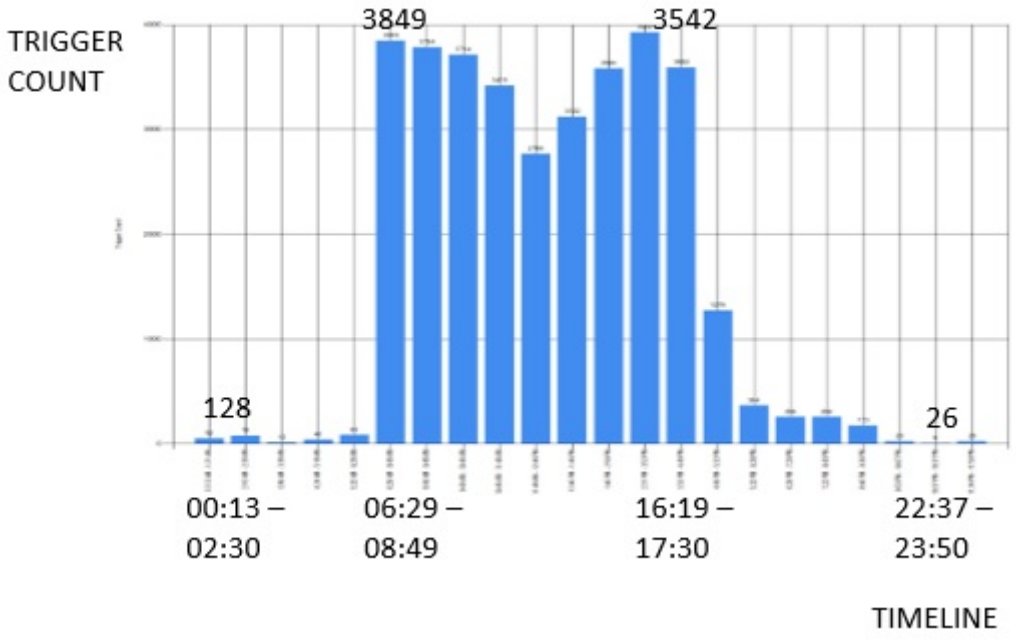


Figure 4.8: Ispec Distribution for INQAL (System 2)

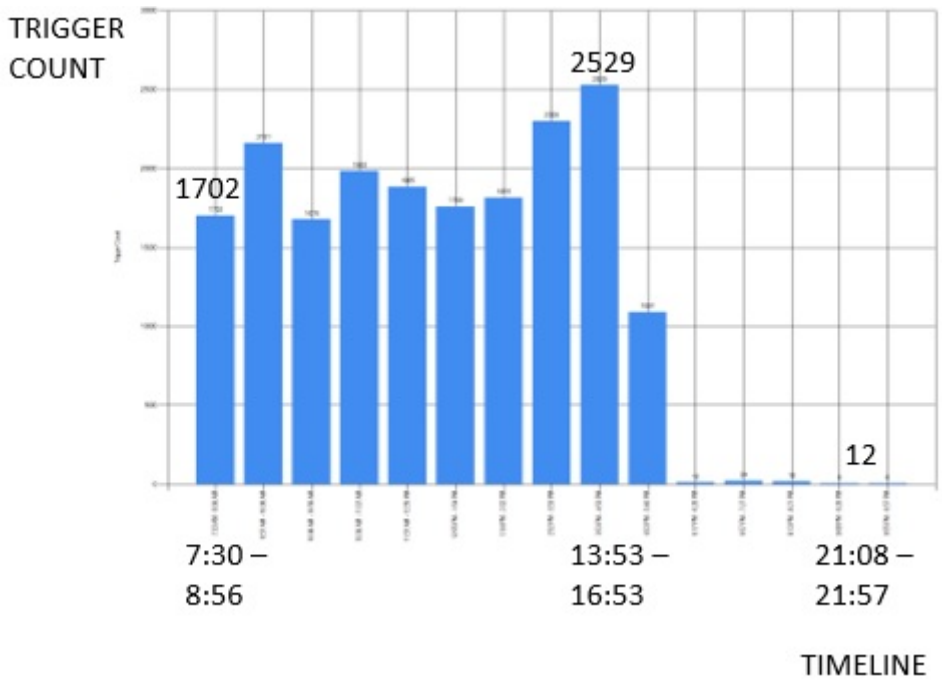


Figure 4.9: Ispec Distribution for CASHI (System 2)

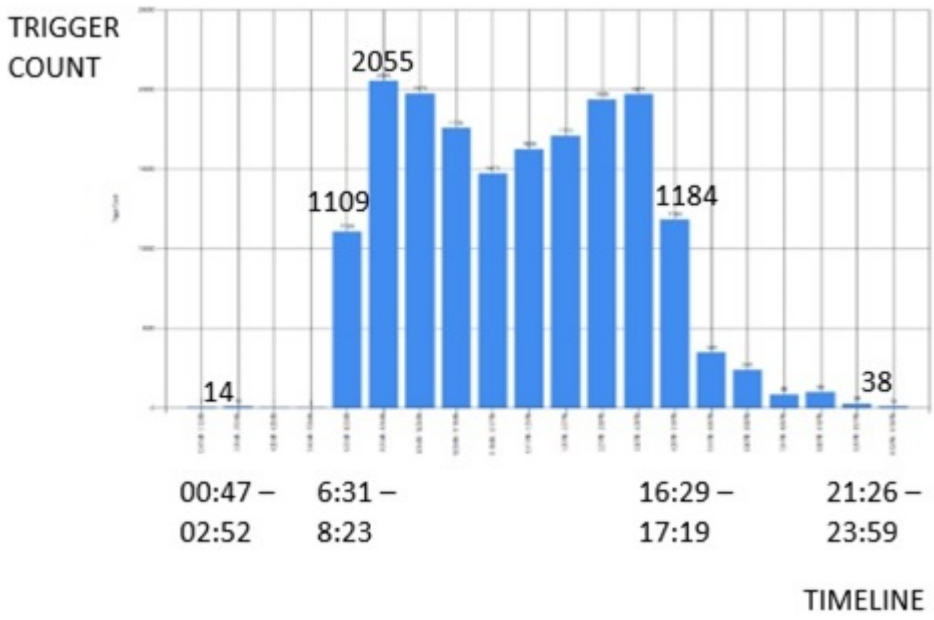


Figure 4.10: Ispec Distribution for WRKOR (System 2)

4.2 Experimental Results and Observations for Response Time Analysis

The algorithm was tested for responsiveness against our two sample applications. Both applications showed similar results as both of them were considered fairly responsive systems. Figure 4.11 shows a snapshot of the results obtained after Response Time Analysis of one of the systems. The results are obtained in descending order i.e. Ispec with larger MTTR value is at the top. Each Ispec is broken down into its constituting process also sorted in descending order. In addition, the number of processes hosting the Ispec transactions and number of transactions used in calculating MTTR is also shown.

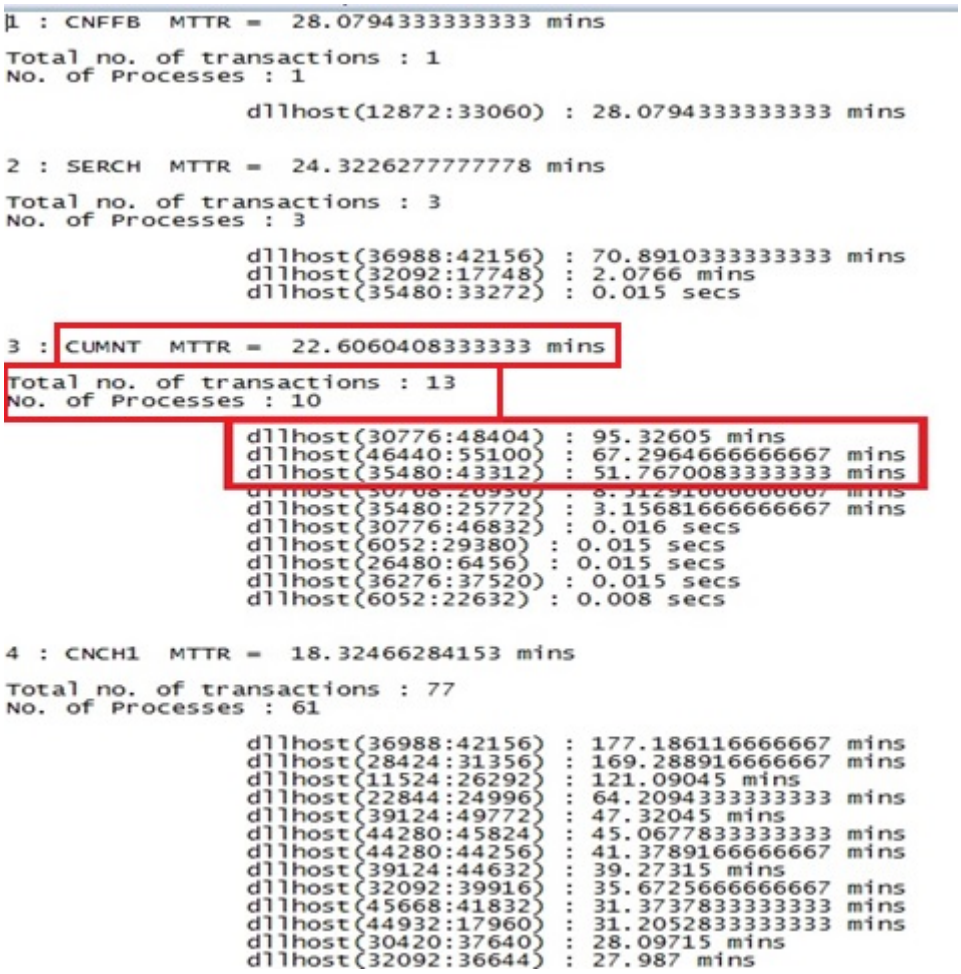


Figure 4.11: A snapshot showing the results of Response Time Analysis

For demonstration purposes we have also compared the Response Time Analysis of our two sample applications whose results are summarized in the table 4.2 below —

Table 4.2: Response Time Analysis for System 1 vs. System 2.

PARAMETERS	**SYSTEM 1**	**SYSTEM 2**
Audit log span (in days)	1	16
Number of files	275	55
Size of audit logs	2.62 GB	537 MB
Total Ispec activity	762760	255991
Number of distinct Ispecs	58	215
Processing Time (in hrs.)	1.75	3.00
Response Times for most frequent Ispecs	INQAL – 1.16 mins **CASHI – 50.75 mins** WRKOR – 2.59 mins CUST1 – 3.26 mins WRKMT – 1.48 mins	SSE15 – 0.52 secs EGE15 – 0.31 secs SSE21 – 2.96 secs SSE73 – 0.03 secs PSE15 – 0.57 secs
Response Times for least frequent Ispecs	VLDIS – 0.046 secs SEQSN – 0.39 secs ISCNW – 0.14 secs RPDFI – 0.45 secs CNSND – 0.11 secs	EGA18 – 0.0065 secs AFE37 – 0.24 secs SSE02 – 0.014 secs SGE02 – 0.024 secs **SSE05 – 1.30 mins**
No. of Ispecs with 0 MTTR	37	1

4.2.1 Observations for Response Time Analysis

We present some interesting observations that follow from Response Time Analysis —

1. HIGH RESPONSE TIMES

   The Response Time Analysis produces an output file which contains the details of Ispecs in descending order of Response Times (MTTR). The Ispecs occurring at the top have larger Response Times. We have seen a trend that these values typically correspond to those Ispecs which are triggered very rarely (relating to transactions that are performed rarely); meaning there count values are very low when compared with the count values of most frequent Ispecs. For example —Table 4.3 below lists the Response Times for the first 16 Ispecs from the output file with the third column representing the count values. The 2 highlighted Ispecs (INQL and CUST3) are frequent Ispecs, i.e. they contribute to 90 % of the systems ispec activity as seen in the audit logs, however their Response Times are fairly large.

   Table 4.3: Ispecs with largest Response Times for System 1.

ISPEC NAME	RESPONE TIME (In mins)	TRIGGER COUNT
CNFFB	28.08	50
SERCH	24.32	515
CUMNT	22.60	245
TLSEL	17.62	94
TLMAS	13.97	82
RNTL1	13.90	48
CNPR1	13.85	991
INQRY	13.14	136
TLLS2	13.12	90
INQSL	**8.59**	**7489**
INQSD	6.67	1064
INTID	4.73	447
LOGON	4.62	3644
CNSVC	4.43	4855
CUST3	**4.09**	**9643**
CASHT	3.91	838

   Hypothesis — Is it safe to say that Ispecs having large Response Times are triggered less frequently and can be safely ruled out as Ispecs that have potentially encountered an exception or a waiting situation? We still have to find out why infrequent Ispecs have large Response Times. and whether this behavior is application or functionality dependent.

2. OCCASIONAL LARGE RESPONSE TIMES FOR A PROCESS

During the analysis we found that for a particular Ispec, there are some contributing processes that have large response times (some more than 40 mins). So, even though the majority of the processes have very small Response Times (approx. 5 millisecond), the introduction of even a single process with a large response time shoots up the overall MTTR value for the ispec. This observation may explain why most frequent Ispecs have fairly large Response Times however, we still need to find out the reason of this abnormal behavior. For Example consider Figure 4.12 below:

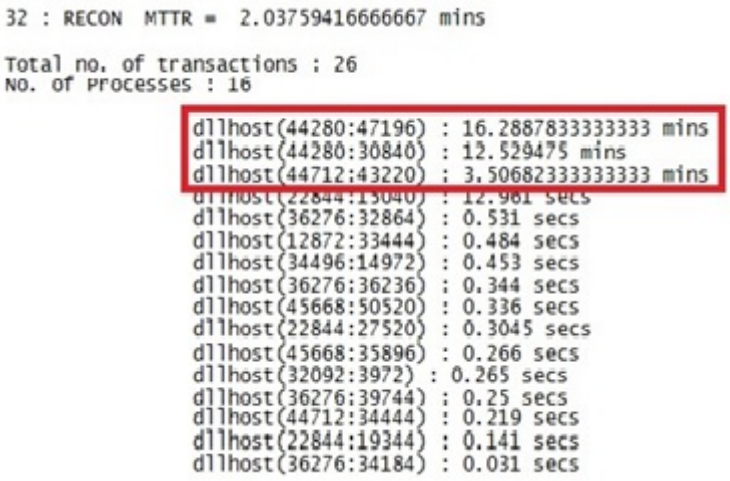


Figure 4.12: A snapshot showing the processes that contribute to increasing Response Time for RECON Ispec

Hypothesis — Could the highlighted processes be responsible for potential faults/exceptions for RECON Ispec? These processes definitely give an indication to potential abnormalities in the system.

3. RESPONSE TIMES FOR FREQUENT ISPECS

Table 4.4 below presents some of the most frequent Ispecs (Ispecs making 90 % of the system activity) of the System 1 with their count values and Response Times - —

Table 4.4: Response Times for most frequent Ispecs of System 1.

ISPEC NAME	RESPONE TIME (In mins)	TRIGGER COUNT
INQAL	1.168	34400
CASHI	**50.75**	**18969**
WRKOR	2.59	17632
CUST1	3.26	14218
WRKMT	1.48	13147
CUST3	4.09	9643
INQWL	1.68	8345
CNSCH	**53.76**	**7542**
INQSL	8.59	7489

Most of the frequent Ispecs have reasonable Response Times —in the range of 5 mins – although it is yet to be verified whether 5 mins is a reasonable estimate for Response Time of an Ispec however, a few frequent Ispecs (like CASHI and CNSCH) have abnormally high Response Times (approx. 50 mins). It is yet to be verified why a particular frequent transaction suffers from such large Response Times.

4.3 Experimental Results and Observations for Exception to Ispec Mapping

As a demonstration of our analysis, the Audit logs and the System logs of two sample ABSuite applications were analyzed. The aim was to find out the Ispecs resulting in exceptions, the total number of exceptions occurred during the span of the System logs, the number and type of exceptions caused by non-Ispec elements and finally calculating MTTF for various Ispecs. This will serve to identify frequently failing Ispecs and the reason for their failures.

Figure 4.13 shows the exception to Ispec mapping obtained for system 1 whereas Figure 4.14 shows the same for System 2. The highlighted entries show an example of a model element to which a particular exception is mapped. A detailed comparison of the two systems is shown in table 4.5 below:

```
EXCEPTION NUMBER : 1
TIMESTAMP : 12/11/2015 6:10:53 PM
PROBABLE ISPECS :
SY001 - IN - D:\Anindya Mukherjea\NIT R\Log Analysis\Logs\Ganesh\Audit Logs\TSIS\Audit_20151211_165212.log

EXCEPTION NUMBER : 2
TIMESTAMP : 12/11/2015 6:11:04 PM
PROBABLE ISPECS :
IV230 - IN - D:\Anindya Mukherjea\NIT R\Log Analysis\Logs\Ganesh\Audit Logs\TSIS\Audit_20151211_165212.log

EXCEPTION NUMBER : 3
TIMESTAMP : 12/11/2015 6:11:12 PM
PROBABLE ISPECS :
SY001 - IN - D:\Anindya Mukherjea\NIT R\Log Analysis\Logs\Ganesh\Audit Logs\TSIS\Audit_20151211_165212.log

EXCEPTION NUMBER : 4
TIMESTAMP : 12/11/2015 6:11:12 PM
PROBABLE ISPECS :
SY001 - IN - D:\Anindya Mukherjea\NIT R\Log Analysis\Logs\Ganesh\Audit Logs\TSIS\Audit_20151211_165212.log

EXCEPTION NUMBER : 5
TIMESTAMP : 12/11/2015 6:11:15 PM
PROBABLE ISPECS :
SY001 - IN - D:\Anindya Mukherjea\NIT R\Log Analysis\Logs\Ganesh\Audit Logs\TSIS\Audit_20151211_165212.log

EXCEPTION NUMBER : 6
TIMESTAMP : 12/11/2015 10:38:23 PM
PROBABLE ISPECS :
Component not an ispec - NA - D:\Anindya Mukherjea\NIT R\Log Analysis\Logs\Ganesh\Audit Logs\TSIS\Audit_20151211_165212.log

EXCEPTION NUMBER : 7
TIMESTAMP : 12/11/2015 10:38:25 PM
PROBABLE ISPECS :
Component not an ispec - NA - D:\Anindya Mukherjea\NIT R\Log Analysis\Logs\Ganesh\Audit Logs\TSIS\Audit_20151211_165212.log
```

Figure 4.13: A snapshot showing different exceptions of System 1 mapped to their elements.

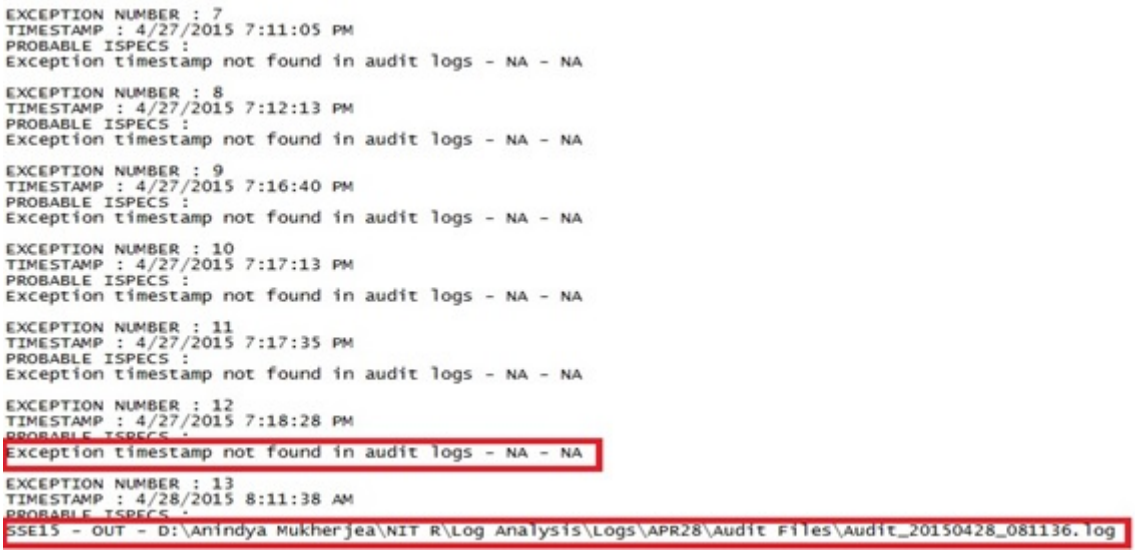


Figure 4.14: A snapshot showing different exceptions of System 2 mapped to their elements.

Table 4.5: Exception to Ispec Mapping results for System 1 and System 2.

PARAMETERS	SYSTEM 1	SYSTEM 2
Audit log span (in days)	1.5	1
No. of files in audit logs	4	275
Total size of audit logs	156 MB	2.62 GB
System log span (in days)	2.5	0.67
No. of files in System logs	5	3
Total size of system logs	195 MB	18 MB
No. of exceptions	54	13
No. of exceptions due to Ispecs	10	1
No. of exceptions due to other elements	36	0
Ispecs mapped to exceptions	SY001 EP711 SA999 SY000 IV230	SSE15
Exception count for each Ispec	4 2 2 1 1	1

4.3.1 Observations for Exception to Ispec Analysis

According to our analysis of the sample applications, we find that most of the exceptions are generated due to non-ispec elements suggesting the fact that Ispecs are not the primary cause of exceptions i.e. user transactions are not responsible for system exceptions. One

possible reason for non-ispec elements like reports generating exceptions is suspected to be their asynchronous nature and simultaneous need for similar system resources.

Chapter 5

Conclusions

ABSuite log files are produced by the ABSuite Runtime during the execution of an application developed using ABSuite. Each log file has its own format and contains valuable information that can be used to enhance performance issues, minimize exceptions, manage system resources better and adopt better design strategies. Our proposed ABSuite Log Analyzer works in three stages. Each stage is responsible for one analysis task. We were successfully able to derive valuable information from the Audit and System logs for two sample applications developed using ABSuite. The two sample applications had some characteristics which were known beforehand like - Usage pattern of System 1 represents typical business hours whereas System 2 is used only for two hours during any day, however in those two hours System 2 is used more heavily than System 1. These characteristics were used to verify the results of our analysis.

The first stage of our analysis finds basic Ispec (or transaction) information. In this stage, we successfully compared and verifed basic Ispec details like trigger counts, distribution of transactions, trigger frequency, usage patterns and etc. for our two sample applications. The information from this stage is used to find out most frequent transactions and usage patterns of the system and transaction distributions over time.

The second stage of our analysis find process level Response times for each transaction of the system. In this stage, we were successfully able to extract Response Times for each Ispec present in both of our sample applications. The analysis showed similar trends for both systems and revealed some observations that were previously unknown. The fact that the overall Response Time depends on the Response Times of the contributing processes gives deeper insight into the analysis stage. Our study enables us to study process level Response Times and isolate the causes of high latencies of a particular transaction.

The third stage of our analysis tries to find out the causes of exception in our system. Exceptions can occur during the runtime of the system due to many reasons like missing DLLs, missing references and files, unavailability of system resources, SQL Server being off-line, processes getting deadlocked and etc. This stage maps various exceptions of our system to the elements that are most likely to have caused them. We were able to test our ABSuite Log Analyzer on two sample applications and were able to successfully identify elements that caused exceptions during runtime. The results of our experiments show that

most of the exceptions are related to Reports and HUB transactions. Only a few transactions actually result in exceptions, however we are yet to verify the severity of each exception. The results of this stage of analysis along with the information of Basic Ispec Analysis stage can help us to predict the time of exception for a particular transaction. This information can be used to reduce system downtime. In case of severe exceptions, appropriate preventive measures can be taken beforehand. This will lead to increase in system availability.

Scope for Further Research

Our work sheds light on the immense information captured in the Audit logs. With proper analysis we can infer more than just usage patterns and transaction distributions. In this thesis we have demonstrated the preliminary stages of ABSuite log analysis. We have found the number of distinct Ispecs that make up a particular system. We have also showed the distribution of Ispecs over time. This work need to be extended to include other non Ispec elements like Reports since these type of elements also play a crucial rule in the proper functioning of the system. We also need to investigate how these elements interact with the ABSuite Runtime to better approximate system performance.

Our thesis also describes the Response times for all Ispecs. As revealed by Response Times Analysis, Ispecs that are infrequent tend to have large response times. This hypothesis needs to be verified further and the cause of this hypothesis needs to be analyzed. It has also been seen that some processes for most of the Ispecs contribute a large value towards the overall Response Time of the Ispec. These processes have to be identified for potential exception causing conditions or any other abnormal behavior. We have to look further into this and analyze the cause of such behavior.

Our ABSuite Log Analyzer is also able to determine the Ispec that causes an exception during runtime by mapping Ispec information from Audit logs to exception information in System logs. Mapping exceptions to Ispecs will help us to create a list of common exceptions that result when a particular transaction is triggered. In the future, this list can also be analyzed to calculate the failure times for all model elements present in the system. This analysis is designed to map exceptions caused due to Ispecs. However, as seen from the System logs, a large number of exceptions are caused by “non Ispec” components like Reports. We have to do a thorough analysis for such elements to get a more accurate analysis. In addition, it is also required to analyze the severity of the generated exceptions. We also need to find out whether we can minimize exceptions by increasing system resources. If so then which resources should be increased and by how much.

Currently our ABSuite Log Analyzer performs the analysis steps as three isolated stages. As part of the bigger picture, we want to develop a tool that will identify major issues and perform the complete analysis by combining results/inferences from multiple stages.

References

[1] M.-S. Chen, J. Han, and P. S. Yu, "Data mining: An overview from database perspective," national Taiwan University, Simon Fraser University and IBM T.J. Research Center.

[2] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Pearson Education, 2006.

[3] J. Valdman, "Log file analysis," Ph.D. dissertation, University of West Bohemia, Czech Republic, July 2001. [Online]. Available: http://www.kiv.zcu.cz/publications/

[4] O. Nasraoui, H. Frigui, A. Joshi, and R. Krishnapuram, "Mining web access logs using relational competitive fuzzy clustering," university of Missouri, University of Memphis, University of Maryland and Colorado School of Mines.

[5] P. Weichbroth, M. Owoc, and M. Pleszkun, "Web user navigation patterns discovery from www server log files," *Federated Conference on Computer Science and Information Systems*, pp. 1207 – 1212, September 2012.

[6] B. Mobasher, R. Cooley, and J. Srivastava, "Automatic personalization based on web usage mining," dePaul University, USA and University of Minnesota, USA.

[7] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa, "Effective personalization based on association rule discovery from web usage data," dePaul University, USA.

[8] R. Ivancsy and I. Vaj, "Frequent pattern mining in web log data," budapest University of Technology and Economics, Hungary.

[9] M. Eirinaki and M. Vazirgiannis, "Web mining for web personalization," athens University of Economics and Business, Greece.

[10] V. Verma, A. K. Verma, and S. S. Bhatia, "Comprehensive analysis of web log files for mining," *International Journal of Computer Science Issues*, vol. 8, no. 3, pp. 199 – 202, November 2011.

[11] Y. Hu, Y. Qian, H. Li, D. Jiang, J. Pei, and Q. Zheng, "Mining query subtopics from search log data," microsoft Research Asia, Xian University and Simon Fraser University.

[12] K. R. Suneetha and R. Krishnamoorthi, "Identifying user behavior by analyzing web server access log file," *International Journal of Computer Science and Network Security*, vol. 9, no. 4, April 2009.

[13] N. Koutsoupia, "Exploring web access logs with correspondence analysis," aristotle University of Thessaloniki , Greece.

[14] C. Shahabi, A. M. Zarkesh, J. Adibi, V. Shah, C. Shahabi, A. M. Zarkesh, J. Adibi, and V. Shah, "Knowledge discovery from users web-page navigation," university of Southern California, USA and Quad Design Technology, USA.

[15] S. Siddiqui and I. Qadri, "Mining web log files for web analytics and usage patterns to improve web organization," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, June 2014.

[16] J. Pei, J. Han, B. Mortazavi-asl, and H. Zhu, "Mining access patterns effciently from web logs," simon Fraser University, Canada.

[17] V. Chitraa and A. S. Davamani, "A survey on preprocessing methods for web usage data," *International Journal of Computer Science and Information Security*, vol. 7, no. 3, 2010.

[18] V. Losarwar and M. Joshi, "Data preprocessing in web usage mining," *Data Preprocessing in Web Usage Mining*, July 2012.

[19] N. M. A. El-Yazeed, "An overview of preprocessing of web log files for web usage mining," port Said University, Egypt.

[20] R. Cooley, B. Mobasher, and J. Srivastava, "Data preparation for mining world wide web browsing patterns," university of Minnesota, USA.

[21] P. Kherwa and J. Nigam, "Data preprocessing: A milestone of web usage mining," *International Journal of Engineering Science and Innovative Technology (IJESIT)*, vol. 4, March 2015.

[22] G. T. Raju and P. S. Satyanarayana, "Knowledge discovery from web usage data: Complete preprocessing methodology," b.M.S. College of Engineering, India and Visvesvaraya Technological University, India.

[23] R. N. Hunter, "Successes and failures of patrons searching the online catalog at a large academic library: A transaction log analysis," 2003.

[24] Z. Lu, N. Xie, and W. J. Wilbur, "Identifying related journals through log analysis," 2009.

[25] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *Proceedings of the 2003 IEEE Workshop on IP Operations and Management*, Tallinn, Estonia, 2003.

[26] T.-T. Y. Lin and D. P. Siewiorek, "Error log analysis: Statistical modeling and heuristic trend analysis," *IEEE Transactions on Reliability*, vol. 39, no. 4, October 1990.

[27] W. Peng, T. Li, and S. Ma, "Mining logs files for data-driven system management," florida International University, USA and IBM T.J. Research Center, USA.

[28] D. Kerr, G. K. W. K. Chung, and M. R. Iseli, "The feasibility of using cluster analysis to examine log data from educational video games," cRESST and University of California, USA.

[29] A. Chuvakin, "Log data mining," 2003.

Index