METHOD TRACE ANALYSER

Video URL: https://www.youtube.com/watch?v=96nsZEiTkmQ

1. What is the uniqueness/novelty added by you to the defined problem statement?

The problem statement is aimed to create an application to aid developers in debugging Java method trace files. We have developed a GUI based application to run and debug a Java application using method trace. It uses SQLite databases to store the trace information, which allows one to run SQL commands over the trace data. It provides intuitive features, which highlights the Longer executing, incomplete and exception throwing methods. It also allows various ways for comparing multiple method traces, such as comparative views and computing difference.

2. How is the proposed solution impacting the business? How are the business processes simplified or bringing value over the existing process?

The application enriches the debugging experience and decrease the time taken for debugging, by exploiting the Method Trace feature of OpenJ9 project, thereby, increasing the productivity of an IT-based business.

3. Architectural flow of the proposed solution, with the mention of technologies to be used in developing the solution.

This section describes the architecture of the proposed solution. It uses the SQLite database, OpenJ9 TraceFormat API and Diff Match Patch Library. The application was designed using Java 8 and to run on OpenJ9 platform. The architecture of the application primarily comprises of 7 tasks – (a) Import Trace, (b) Run application with Trace, (c) Show Trace Information, (d) Show Method Tree, (e) Show Difference, (f) Run SQL Command on Trace Data. Fig. - 3.1 describes the architecture of the proposed solution.

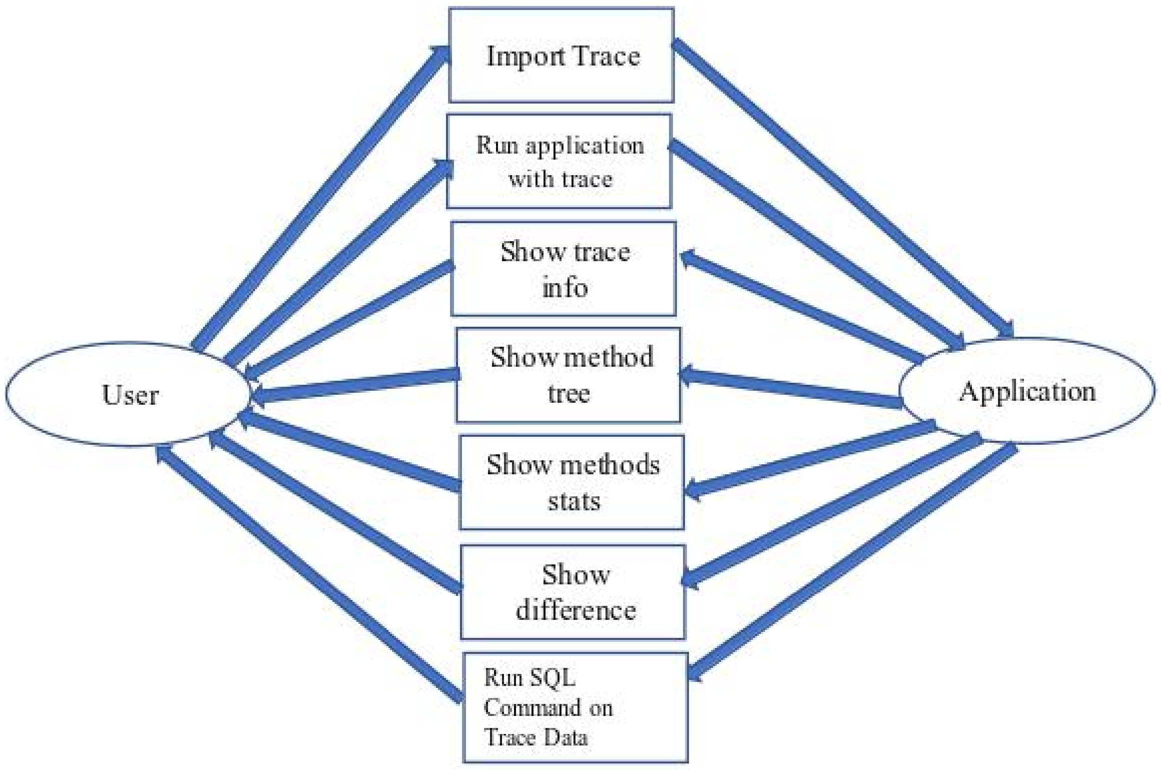

Fig. - 3.1 The Data Flow Diagram of the solution.

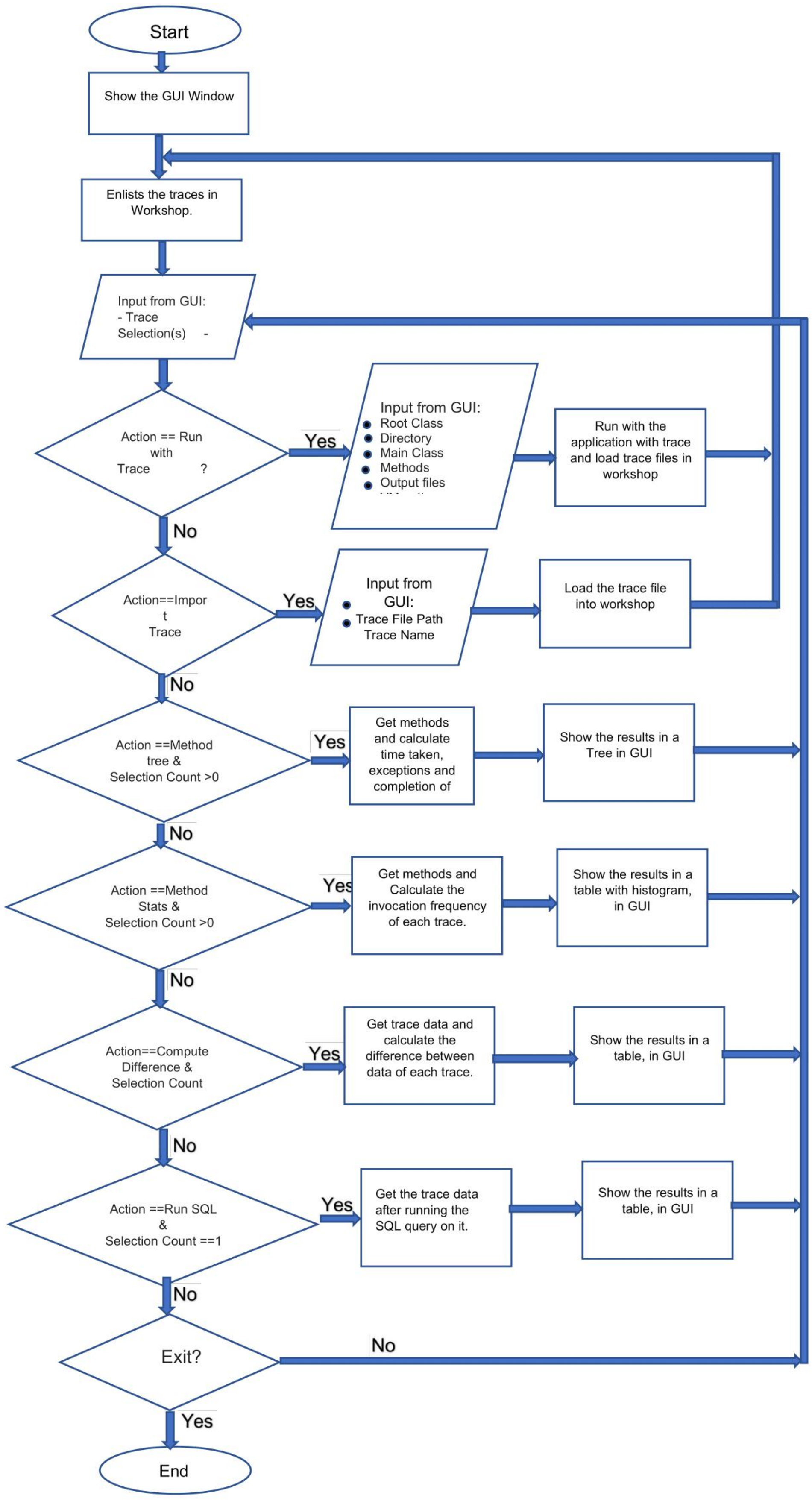
Start
Show the GUI Window
Enlists the traces in Workshop.
Input from GUI: - Trace Selection(s) -
Action == Run with Trace ?
Yes
Input from GUI:
Root Class
Directory
Main Class
Methods
Output files
Run with the application with trace and load trace files in workshop
No
Action==Import Trace
Yes
Input from GUI:
Trace File Path
Trace Name
Load the trace file into workshop
No
Action ==Method tree & Selection Count >0
Yes
Get methods and calculate time taken, exceptions and completion of
Show the results in a Tree in GUI
No
Action ==Method Stats & Selection Count >0
Yes
Get methods and Calculate the invocation frequency of each trace.
Show the results in a table with histogram, in GUI
No
Action==Compute Difference & Selection Count
Yes
Get trace data and calculate the difference between data of each trace.
Show the results in a table, in GUI
No
Action ==Run SQL & Selection Count ==1
Yes
Get the trace data after running the SQL query on it.
Show the results in a table, in GUI
No
Exit?
No
Yes
End

Fig – 3.2 The Flow Chart of the Solution

4. Define the scope of work to be implemented in the project with modules etc.

The scope of work of the application will include implementation of the processes of - (a) Comparing two trace files: one from failing and passing case each and to find out the anomaly. (b) Parsing one or more trace file and suggest anomalies - i.e. Flag methods which are not completing their execution /

taking longer to execute (c) Parsing one or more trace file and create a tabular and graphical view for the number of times each method is invoked. (d) Comparative view in case of multiple files. (e) Show the results using GUI.

The solution to the problem can be broken down into 7 modules – (a) Import Trace, (b) Run application with Trace, (c) Show Trace Information, (d) Show Method Tree, (e) Show Difference, (f) Run SQL Command on Trace Data.