International Journal of Engineering & Technology, 7 (4.10) (2018) 371-374

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper



Trace Crawler

E Umamahewari[1]*, Krishnaveni S[2], Xiao-Zhi Gao[3]

[1]School of Computing Science and Engineering, VIT University, Chennai, India.
[2]Assistant Professor, SRMIST, Kattankulathur, Chennai
[3]Lappeenranta University of Technology, FInland
**Corresponding author E-mail: umamaheswari.e@vit.ac.in*

Abstract

The Object Request Broker (ORB) manages the interaction between clients and servers using the Internet InterORB Protocol (IIOP). The ORB trace file comprises of messages, trace points, and wire tracing. These trace files are usually large and at times they can grow to 1-2 GBs. For debugging an ORB problem, the developer must determine whether the problem is in the client or in the server of the distributed application. Typically during debugging process, developer have to switch between source codes and trace logs back and forth multiple times. Often browsing the right source code file, opening it in the IDE and then moving to the line from where the trace is originated. This work proposes a tool which will help developers and service engineers to browse through source code and corresponding trace by providing integrated framework. The Aim of the tool is to build an application to parse and analyse ORB trace files. The application will fetch relevant details for the failing entries, and will perform the preliminary analysis. In this way a lot of time and human effort can be saved. The customers such as defense, military organizations can themselves view relevant details about the failures of their application by this tool, without having to compromise on confidentiality with minimal effort and time.

Keywords: *Object Management Group, CORBA - Common Object Request Broker Architecture, IDE- Integrated development environment, IIOP - Internet InterORB Protocol, Eclipse Plugin.*

1. Introduction

T CORBA the leading middleware solution standard was developed by OMG for providing interoperability among distributed objects. CORBA enables the exchange of information, independent of Operating systems, hardware platforms, and programming languages [1] [2]. Essentially CORBA is a design specification for ORB, where ORB implements the mechanism required for distributed objects to communicate with local machines or remote machines, which were developed using different languages and different locations in a network [3]. Traces are generated when communication happens between client and server [4]. These traces help in understanding ORB [5].

Pattern Matching Java: Java supports pattern matching via its character and assorted string classes. Because that low-level support commonly leads to complex pattern-matching code, Java also offers regular expressions to help write simpler code. A regular expression, also known as a regex or regexp, is a string whose pattern (template) describes a set of strings. The pattern determines what strings belong to the set, and consists of literal characters and metacharacters, characters that have special meaning instead of a literal meaning. Java's java.util.regex package supports pattern matching via its Pattern, Matcher, and PatternSyntaxException classes.

Eclipse Plugin Development**:** A software component in Eclipse is called a plug-in. The Eclipse platform allows the developer to extend Eclipse applications like the Eclipse IDE with additional functionalities via plug-ins [6]. Eclipse application uses a runtime based on a specification called OSGi. A software component in OSGi is called a bundle. An OSGi bundle is an Eclipse plug-in.

ORB is an object Bus which enables objects to make transparent request and send response to other objects locally and remotely even on a different platform [7] [8]. ORB is an implementation of CORBA and IIOP specs. An ORB uses CORBA Interface Repository to locate and communicate with a requested component. ORB achieves this inter-operability by assuring everything is in IDL [9]. A programmer while creating a component uses CORBA's Interface Definition Language (IDL) to declare its public interfaces or the compiler of the programming language to translate the language statements into appropriate IDL statements. The statements are then stored in the Interface Repository as metadata. Applications which require simpler functionality can avoid the complexity of CORBA and go with SOAP or REST. Currently ORB is widely used in implementing Java EE. Standalone CORBA programs (Either in Java or C++ or other languages) can easily communicate to the Application servers like (WebSphere and WebLogic [10]. While debugging ORB often involves browsing to the right source code file, opening it in the IDE, moving to the line from where the trace is originated, switch between code and logs back forth multiple times, depending upon the problem the trace logs can be large. This is cumbersome, frustrating and time taking process. It is necessary for the service engineer to visualize the execution path in the code to understand what is happening instead of manually going through all the trace points and thread related information. Fetching relevant details for the failing entries, and acting on the defect is a time taking process. IOR are usually represented as Strings, they contain the hostname and port of the Object. Apart from this they also store ORB profiles. ORB profiles contains a variety of information ranging from ORB's major and minor version (Helpful in identifying the features supported by the other ORB.). To identify an object, the server uses an object key that is opaque to anybody except the hosting server. The

Copyright © 2018 Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

client obtains object key from object reference. The IOR and Object Keys are hex data that is needed to be decoded. Manual Assumptions may turn wrong in these cases so automating this will help service engineer save their time and effort.
The log files grow large in size (1-2 GB) and it is hard to analyse without a tool. In case of any urgent defect it will be hard to analyse the file all at once. If the engineer is new to ORB and wants to know how it actually works it's hard for them to navigate from the trace log to source to know its exact functionality. So, having a UI which will directly lead to source code will be beneficial. Navigating to source code directly will clearly state the problem of the defect and will take less time for the engineer to analyse the problem. Apart from all these instead of reading line by line to find the problem a preliminary analysis can be done to know the exact problem, so that it takes less time for the engineer to turn down the defect.
The main aim of this work is to develop a one point ORB trace analysis tool. The tool should be developed in such a way that the results obtained are accurate with minimal work load. . Obtaining relevant data based upon the exceptions occurred and providing a clear view to the Service Engineer is also focused in this work. To integrate the trace file with IDE an eclipse plugin is used. This plugin creates view where ORB traces can be viewed and pattern matching is used to get relevant data and fetch information according the line. These data will help them in finding the cause of defect quickly than usual. The hex data should be interpreted correctly and displayed when selected.
Trace Crawler will help Service Engineer to visualize the execution path in the code to understand what is happening, instead of manually going through all the trace points and thread related information. It will be useful for service engineers, developers involved in WAS, ORB and similar products/applications. These machine learning should be done in cloud so that to use more storage and process resources and store the patterns [11] [12]. But handling the security issues of cloud should be a challenge [13]. The cloud platform should ensure both the privacy and security of the data [14].
Section 2: The detailed description of the proposed system is explained with the objectives, hardware and software requirements.
Section 3: This section deals with the proposed system architecture explained using UML diagrams
Section 4: This section discusses the advantages of the proposed system over the existing system with experimental results.
Section 5: This section discusses about the process of implementation of the proposed system.

2. Proposed System

Aim: Making it a convenient and ease of access as its one point ORB trace analysis tool.
Objective(s)

- Integrating the trace file to IDE. Making the hex data readable.
- Provide relevant information for defect fixing. Analysing large log files.
- Exception based Information gathering.

Hardware Requirements: Hardware - Pentium Speed - 1.1 GHz, RAM - 8 GB, Hard Disk - 20 GB Eclipse Memory- 8GBs
Software Requirements: Operating System: Windows Family, Linux, Technology: Java and J2EE, Eclipse IDE: Any version, Java Version: IBM Java 8
Trace Crawler for ORB traces is flexibly designed. The Trace Crawler is developed in such a way that it is easily understandable to programmers and functionalities developed can be reused with ease. Figure 1: representation view about the features of Web Sphere Application.
An eclipse based plugin is developed as the ORB source code is in java and most of the developers use eclipse as they can use deploy their own plugins in it and use. So, based upon the requirements, a separate view for the trace logs in the same IDE is created.

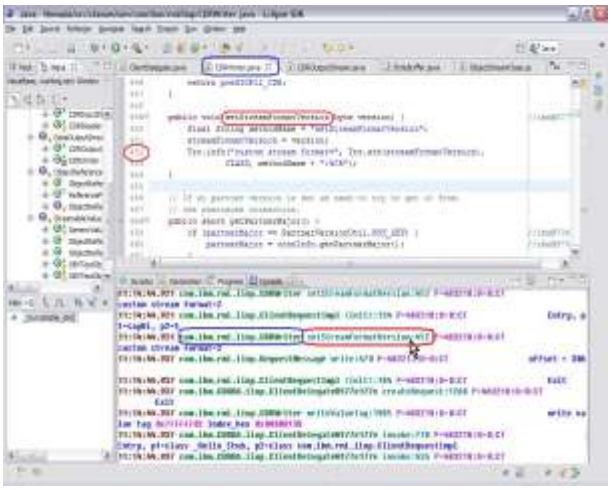


Fig. 1: WEBSPHERE Application

An eclipse plugin is developed with a view, is developed where the trace file will be loaded. The complete file cannot be loaded all at once as JVM cannot load the file all at once, a buffered reading is used to read few lines at a time, in a way to overcome the problem of heap dump. So, to redirect the traces to source code a selection listener is added to the view which will take it to the eclipse editor. Based upon the line selected it will use pattern matching to find whether it is a hex data or trace line and based upon that it will either show the decoded hex data or lead to the source code. Based upon the exceptions in the log file the related data have to be captured i.e. in case of communication failure the frequently failing IP/port combinations and in case of marshal exception the thread related information is captured.

3. System Architecture

This eclipse based plugin, revolves inside eclipse IDE. In this plugin, the trace file and the project name from the current workspace is loaded as the input. Two views are created in the eclipse IDE, One is the trace view where the entire trace file will be loaded. This view will direct the developer to the source code of the trace selected in the view. Apart from moving to the source file words can be searched, like the general search operation. When the hex data is clicked the plugin identifies what kind of data is selected and when clicked, it will produce the relevant details.
Apart from all this an analysis can be done on the file and necessary details are obtained from the trace log. According to the exceptions present in the file, data is obtained and these data is collected. The Figure 2: Represents the sequence of the entire system. This depicts the flow of the entire system.



Fig. 2: Sequence-Admin

4. Process of Implementation

The project is developed in Java Programming Language by using the Eclipse Juno Integrated Development Environment (IDE). The Java Development Kit (JDK) is used which includes a variety of custom tools that helps to develop plugins.
Plugin Development: Developing an eclipse plugin as it the most commonly used IDE among ORB developers. This plugin takes the trace log and project name (from the current workspace) as input. Provides us a view with trace log from where we can navigate to the source code of the traces. Presenting the hex data in a more readable format.
Filter View: In this view we get the traces for a particular exception. Thread and other related information of that exception is represented in this module.
Post Processing Data: Collecting the data based upon the exceptions in that given file. Representing the data in an understandable format. Helping the engineer to find out the problem in the defect.
Experimentation: Crawler is implemented in Java and evaluated with the experts in ORB to verify whether it is capturing the correct data and redirecting to the correct source file. To evaluate the performance of the tool, we can just see how quickly the file is loaded and how file operations are happening. Handling a large data is a big task and still some work needs to be done on it. While handling the file operations we need to look at various techniques to minimize the load in RAM. So loading the file completely into memory will be a problem.
Optimization: To improve the performance of file operations a strategy is being proposed such that the file will not be loaded into memory and the file operations will be done directly to the file without any memory wastage. This code is further optimized in such a way

that it can be used for any other log files. As this is going to be an open source plugin changes can be done depending upon the requirements and used for other log file. This code is so generic and making changes to it is easy as only the patterns have to be changed.
Evaluation**:** A correctness evaluation of this crawler is performed by ORB experts. The goals include: evaluating the efficiency of Crawler in pattern matching, analysing the data from file, and capturing the accurate data. More than performance data correctness is the major evaluation for this project. As it all depends on the data captured. These data is used in defect fixes if this data is wrong then it will be waste of time for the engineer to find out what went wrong.
Screen Shots**:** Figure 3 & 4 represents the screen shots of the developed tool. Figure 3 represents the filter view with marshal exception and Figure 4 represents the post processing data details.

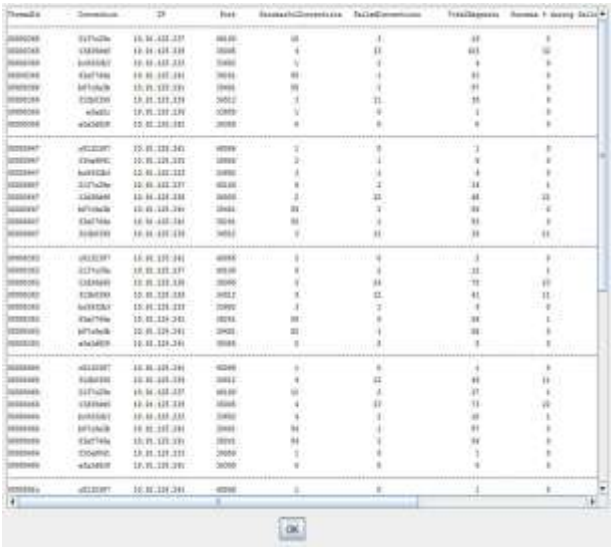


Fig. 3: Filter view with marshal exception.



Fig. 4: Post processing data details.

5. Advantages of Trace Crawler

This project is application development to automate things in the current digital technology. This helps in reducing the workload of engineer. Computers are faster and accurate than humans so the accurate results will be produced through this. The cumbersome process is replaced with an interesting plugin which is easy to install and use. It saves around 15min per defect and decreases the turnaround times too. This can be made useful for other trace files apart from ORB and is user friendly.
Organizational Feasibility: The application would contribute to the overall objectives of the organization. It would provide a quick, automatic and cost effective solution to the log files. It would provide a solution to many kinds of log files which needs a look up to the source code. As the new system is flexible and scalable it can also be upgraded and extended to meet other complex requirements which may arise in the future.
Economic Feasibility: The project is economically feasible as it only requires a system with a Windows or Linux operating system. The application needs an eclipse IDE and this plugin can be added to it. The users should be able to understand how to give input and navigate through files in the view.
Technical Feasibility: To develop this application, an eclipse IDE with 4GB memory allocated to it is needed and the user is required to have knowledge with the eclipse plugins and an understanding of ORB is required. The current project is technically feasible as the plugin was successfully deployed in an eclipse IDE having Windows 7 operating system.
Behavioral Feasibility: The application is behavioral feasible since it requires no technical guidance, all the modules are user friendly and execute in a manner they were designed to.

6. Results and Discussion

This approach achieves minimizing the work load of an engineer and automating the existing manual work. As computer work is more efficient and accurate compared to human work this saves a lot of time and effort. This increases the understanding over ORB through the navigation.
Performance Metrics:
Response Time: The response of the application is accurate. As it takes time to load to file and to display, it takes a few secs. In these few seconds, we cannot perform any operation in eclipse IDE as is the UI is not responsive.
Storage: The Eclipse IDE is allocated a memory of 4GB as this memory is further used to allocate the file space. The files should be loaded into memory and perform some operations. Large amount of memory is allocated as these trace logs are large in size and to minimize this memory we are further going to use _le operations so that the memory will not be affected.

Existing System - Proposed System Comparison: This is totally a new system. The manual analysis is automated and this can be useful for the types of trace logs generated. This automation reduces human effort and time. This will help the developer or engineer to understand the actual working of an ORB or any other system it wants to use. As navigating is already a part of java. Navigating through trace files will be an addition to it. Synthesis of process: It is challenging to map the patterns according to requirements. As pattern matching is mostly accurate but can have mismatches. The data to be captured should be very accurate to have better pattern matching. Machine Learning plays a vital role, since each step of the project will depend on the previously captured data. In order to capture the data, the data has to be stored in an efficient way. So the data is stored through lists and maps. These machine learning should be done in cloud so that to use more storage and process resources and store the patterns. These lists and maps are used to store the last searched words and their index, as to reduce the turnaround times and perform word wraps.

7. Conclusion and Future Work

A system is proposed that will help in leading to source code through trace logs. This is going to be a generic system which can be used by any log file. This is going to be an open source plugin so that those who need can change the code accordingly. This is an automation to reduce manual effort, as automating things will lead to minimizing the time and effort to fix a defect. Further this plugin can be changed according to the trace logs and will help the developers or engineers in debugging process. As of now the file is stored in the RAM and operations are performed on it. We can make the machine learn the common failing patterns and detect the pattern, this will help to identify the defect in one go. These machine learning can be done in cloud, so that more storage, and process resources will be available to store patterns.

References

[1] What is corba? http://www.corba.org/

[2] Overview of CORBAhttp://www.cs.wustl.edu/~schmidt/corba-overview.html

[3] CORBA::ORB Class Reference-http://www.dre.vanderbilt.edu/Doxygen/6.0.1/html/libtao-doc/a00122.html

[4] Common Object Request Broker Architecture (CORBA) https://www.ibm.com/support/knowledgecenter/en/SSMKHH_10.0.0/com.ibm.etools.mft.doc/bc22400_.htm

[5] Common Object Request Broker Architecture-https://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture. Last Viewed: 14-5-2017

[6] Eclipse Plugin Development-menthttps://www.packtpub.com/mapt/book/application-development/9781782160328/1. Last Viewed: 14-5-2017.

[7] The Object Request Broker (ORB) Architecture-http://www.nyu.edu/classes/jcf/g22.3033-007/handouts/g22_3033_011_h41.htm

[8] ORB Basicshttp://www.omg.org/gettingstarted/orb_basics.htm

[9] Object request broker-kerhttps://en.wikipedia.org/wiki/Object_request_broker

[10] CORBA ORBhttps://docs.oracle.com/cd/E13211_01/wle/wle50/jref/jprorb.htm

[11] D.M. Ajay and E. Umamaheswari, "An Initiation for Testing the Security of a Cloud Service Provider", Smart Innovation, Systems, Technologies, Springer Publications 2016, Pg. No 35-41.

[12] D M Ajay, Umamaheswari .E, "Why, How Cloud Computing – How Not, and Cloud Security Issues", Global Journal of Pure and Applied Mathematics (GJPAM), Volume 12, Number 1, 2016.

[13] Umamaheswari E, Ajay DM, Umang Sindal, "Scope of Internet of Things: A Survey", Asian Journal of Pharmaceutical and Clinical Research, April 2017.

[14] D.M. Ajay, Umamaheswari E, "Evaluating the Efficiency of Security Mechanisms in Cloud Environments", International Journal of Control Theory and Applications, Vol.9, No.51, 2016.