ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 7, July 2015

An Efficient Automatic Debugging System for Object Oriented Approach

Ashvini A. Patil, Prof. M.C. Kshirsagar

ME Student, Dept of Computer Engineering, Vishwabharti COE, Pune University, Ahmednagar, Maharashtra, India

Assistant Professor, Dept of Computer Engineering, Vishwabharti COE, Pune University, Ahmednagar, Maharashtra, India

ABSTRACT: In the process of Software Development and evolution, Developer has to answer multiple questions about how the code or software behaves at runtime. The traditional or classical debugger while debugging gives developer bunch of breakpoints in the source code.

Object based debugging offer, interruption when a given or a particular object is accessed or modified. Programmers, who try to find violations in such source code, need new tool that allows them to explore objects in the system effectively. The implementation of the proposed debugging actually offers programmers an effective tool which will allows searching of objects even for programs that have huge number of objects.
As stated from traditional tools that, the complexity of object oriented system increases, debugging becomes relatively difficult. Developer needs a dedicated user interface for these operations on objects; this need is fulfilled by facilitating a user interface for the programmer.

Object based debugging tool looks forward to analyse the relationship in between the objects during the runtime. So the key behind this is to focus on a particular object instead of the execution stack. This allows functioning operations directly on objects rather than on the execution stack. The presented tool can allow user or a developer, different operations, which going to perform on a particular object. There exists therefore conceptual gap between the interface offered by the debugger and the need of the developer, hence to overcome or fill the gap; there is a need for object based debugger and useful interface for it.

KEYWORDS: Software programming, Debugging, Error, UI, Bugs.

I. Introduction

Classical Debugger concentrated on the execution of stack. The programmer recognises parts of source code of interest and sets breakpoints accordingly. These breakpoints are set of purely with respect to static abstraction and not respect to particular object of the running system. Object based debugging as an alternative approach to interacting with a running software system. Focusing on objects as the key point, natural debugging operations can be defined to answer developer questions related to runtime behaviour. Here, presented scenario of an object based debugger. How it offers more effective support for many typical developer tasks than a traditional or classical debugger.

Classical debuggers are not always up to the task, since they only provide access to information that is still in the run-time stack. In some cases, the information needed to track down these difficult bugs content; how an object reference got here, the previous values of object's fields.
For this reason it is helpful to have previous object states and object reference flow information at hand during debugging.

II. Related work

For proposed work to be better one following literature is analysis for existing systems working and critically evaluated on some evaluation method to find shortcomings from them.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 7, July 2015

Back-in-time debuggers approach.

These are extremely useful tools for identifying the causes of bugs. Compare to the "omniscient" approaches that try to remember all previous states are impractical because they consume too much space or they are too slow. So many approaches to limit these penalties, but they ultimately end up giving out too much relevant information.

In this paper a practical approach that attempts to keep track of only the relevant data. In contrast to other approaches, it keeps object history information together with the regular objects in the application memory. This method has the effect that data not reachable from current application objects that's why not useful further.

This approach, present idea which explains that memory utilization stays in practical limits. Furthermore, the performance penalty is significantly less than with other approaches [1].

Back-in-Time Debugging:

Back-in-Time Debuggers are useful tool for identifying the cause of errors, not the omniscient debugger which always remembers all previous states.

To overcome this drawback of omniscient debugger back in time debugger is developed. Omniscient Debugging: also known as back-in-time debugging or reversible debugging.

These debuggers store the total history and execution trace of a debugged program. Developers can explore the history by simulating step-by-step execution both forward and backward [1] [6].

Query Based debugging approach.

User defines a query in a higher-level language that is then applied to the data Queries can test complex object interrelationships and sequences of related events.

Trace oriented Debugger: it is collected of a well-organized instrumentation for incident making, a specific database for scalable storage space, and support for partial traces to reduce trace volume [2].

While this method has the advantage that nowhere data is lost, its drawback is that it requires large hardware power, which is not available for many developers today [6].

The why line debugging interface approach.

Why line tool which facilitate developer to ask, "Why did" and "Why did not" questions regarding their program's output Why line tries to facilitate developer by applying static as well as dynamic analyses and after that answer Some of the developer questions [7].

Auto Flow an automatic debugging approach.

Aspect-oriented programming (AOP) is gaining popularity with adoption of languages such as AspectJ. During AspectJ software evolution, when tests fail, it may be lengthy or difficult for programmers to find out the failure minimising changes by manually inspecting all code editing.

To beat the costly attempt spent on debugging developed AutoFlow, an automatic debugging approach for AspectJ system. AutoFlow meets the potential of delta debugging algorithm with the benefit of change impact analysis to slow down the search for imperfect changes. It primary uses change collision analysis to identify a subset of responsible changes for a failed test, after this ranks these changes according to proposed heuristic (indicating the likelihood that they may have contributed to the failure), finally this improved delta debugging algorithm to determine a minimal set of faulty changes.

The important advantage of AutoFlow is that it can automatically reduce a big portion of irrelevant change in an early stage, eventually then locate not fixed changes effectively [8].

How helpful are automated debugging tools :

The Area of automated debugging, which is with the automation of identifying and correcting a failure's root cause, made tremendous advancements in the past years. However, some of the reported progress may be due to unrealistic assumptions that with the evaluation of automated debugging tools.

These unrealistic assumptions concern the work process of developers and their ability to detect wrong code without explanatory context, or the size and arrangement of fixes. Instead of trying to locate the fault, this proposes to help the developer understand it, thus enabling her to decide which fix they deems most appropriate.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 7, July 2015

This came to know the need to employ a completely different evaluation scheme that bases on feedback from actual users of the tools in realistic usage scenarios [9].

NUDA a Non-Uniform Debugging approach.

This paper is proposed a novel non-uniform debugging architecture (NUDA). This makes hardware-assisted debugging both feasible and scalable for many-core processing scenarios. Here, theme is to distribute the debugging support structures across a set of hierarchical clusters while avoiding address overlap.

It allows the address space to be monitored using non-uniform protocols and propose approach to lockset-based race detection supported by the NUDA. Here, page-based monitoring cache in every NUDA node to keep track of footprints. The union of all the caches know how to take in account as a race detection probe without violating execution ordering. [10].

"A Review of reverse debugging"

Reverse debugging is defined as of a debugger to stop after a failure in a program has been observed and go back into the history of the execution to find reason for the failure.

Reverse execution has become a practical technique available in a number of free and commercial tools. This article review the history and techniques of reverse debugging, as researched, implemented, and used until today [11].

There is a need to find or steer in area where programmers actually face problems during debugging scenario [12]. This strategy works well, trying to understand the general performance for objects. When addressing polymorphism or delegation the performance of objects of same class changes on their composition. In these scenarios need an object-specified analysis and simple breakpoint strategy is not the best option. In application development when programmers require interrupting the execution of the application when a particular code is evaluated, requires breakpoint strategy. The programmer wants to locate the particular object he is concerned. The programmer specifies a suitable condition to recognize the particular object previously found, without interacting with it. This approach may be practicable, if exist few objects to analyze in given code [13].

2.2 Related work shortcoming

Studding and analysing different literature survey following are the outcomes.

- ✓ Back in time debugging debugger have to remember history of all previous states.
- ✓ Trace oriented debugger requires more hardware power, which is practically not possible. Omniscient debugger depend on more memory because, to store history of last stages. Reverse debugging is to stop after a failure in a program has been observed and go back into the history of the execution to uncover the reason for the failure.
- ✓ AutoFlow can automatically reduce a large portion of irrelevant change in an early phase, eventually then locate faulty changes effectively.
- ✓ After going through literature survey came to know that developer faced some kind of problems while doing debugging.
- ✓ Major problem is that developer cannot answers about objects.
- ✓ After taking view on problems faced by developer they do not get answer to their question regarding object.
- ✓ There is pretty need of a useful and dedicated user interface for debugging scenario.
- ✓ Developer comfortable with using object oriented dedicated user interface for debug situations.

When complex object oriented system taken in account then traditional debuggers fails to act on object related operations and relationship between different objects.

To eliminate these problems new tool should be developed on object based approach and useful dedicated user interface for it.

III. Motivation Scenario

The motivation for doing this project was primarily an interest in undertaking a challenging project in an interesting area of debugging. This gives opportunity to learn about new area of software engineering. This area is possibly an area that I might study at postgraduate level. As the debugging area taken into account developer came across different problems, which are faced by developer.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 7, July 2015

The traditional debugging technique used by programmer is concentrated on stack orientation so developer face problems regarding objects in the code given. The debuggers not designed to answer many of the questions that developer typically uses to ask after analysing different papers related to approaches of debugging, found that one can develop a debugging tool which is based on objects, and possesses following some points to understand runtime behaviour of the system.

It will be helpful to continue interacting with the runtime, applying operations directly to objects without working with static representation of the system. This is useful in to monitor communications with entity objects without taking stepwise breakpoints.

So it is required to develop object based debugging tool that facilitated with user interface which fulfil needs of developer such as, different interruption related to objects or keep watch on object interactions and do operations related to objects using user interface telling suggestions.

3.1 System Description

Looking on problems faced by user or developer they do not get answer to their question regarding object. When complex object oriented system taken in account then traditional debuggers fails to act on object related operations and relationship between different objects. To overcome this object based debugging tool is very helpful in this scenario. In this tool brifost reflection framework is being used. The tool of object based debugging is built on top of the Bifrost reflection framework. Bifrost offers fine grained unanticipated dynamic structural and reflection through meta-objects. Instead of providing different reflective capabilities as an external mechanism integrate all deeply into the environment. Explicit meta objects providing a range of features, thereby evolving both application models and the host language. Meta-objects provide a sound basis for different coexisting meta-level architectures by giving traditional object-oriented techniques to the meta-level.

3.2 System Workflow

System workflow of object based debugging have following steps in the system workflow.

1. Take source code into object based debugging.
2. Object based debugger find particular needed object.
3. It finds relationship with other objects.
4. Searching objects from huge source code.
5. Developer now acts on object.
6. Using user interface user do different operation on object
7. We can use different keyword for getting information about debugging of any input program like info, help, abort, retry, throw..
8. Apply this procedure repetitively on whole Source Code document for desired objects.
9. Object related operations performed. If any error or bug come you can use throw functionality for getting more details about it.
10. Make changes in objects and press retry functionality for executing it.
11. Prevent problems and so improve performance.

3.3 System Overview

The source code when debug using object based debugging tool, particular object required by developer is searched and made available to developer. Developer further acting on object do the specified operation by using user interface concentrated on objects. The code file taken into proposed tool, then code parsing done for all particular objects. After going through execution and isolates the points needed by developer needs.

The parser extracted all objects from provided code file then supplied or given to execution module. This parser also converted it into intermediate forms which give response to object related errors or bugs. In code generating module there is code which gives object related error findings. Finally execution step it operates on the code parsed taking objects in consideration using a dedicated useful interface for it. The stepwise execution is stated in system workflow

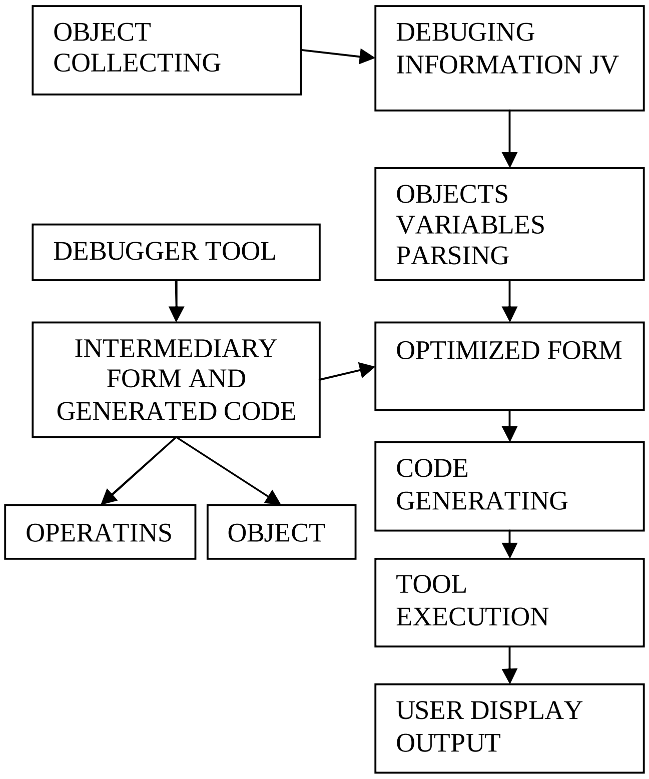


Figure: 1 overview for object based debugging system

IV. PERFORMANCE RESULTS

Various experiments and result were conducted for calculating performance of this prototype. Existing system is compared with this developed prototype in terms of time usage and memory usage.

Developed prototype and Netbeans debugger is compared with three systems in terms of time consumed and their respective memory usage. Following table shows comparison in between Netbeans Debugger and developed prototype.

Input system	**NB**	**Bug**	**Developed Prototype**	**Bug**
Introduction Example	472024	1	344976	3
Clock System	294256	2	290466	2
Moving Button System	562574	1	260130	2

Table 1: Results Analysis

Those systems shows memory required for execution. In netbeans (for example) debugger execution done without considering the memory required for their GUI while in developed prototype during execution memory required for GUI is also considered, that memory is in 100000 bytes.

So here in case of those systems we have minus these bytes from results as shown in Table 1, time is measured in milliseconds while memory measured in bytes.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 7, July 2015

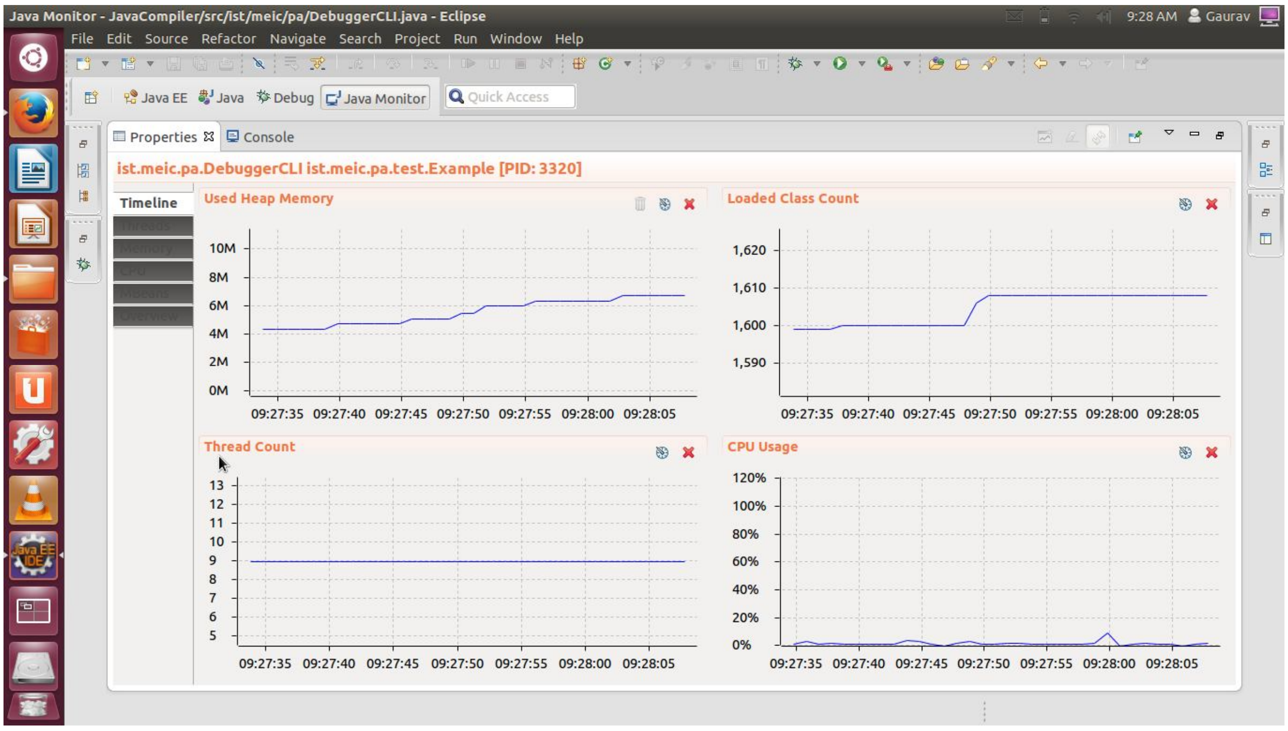
Input System	NB	Bug	Developed Prototype	Bug
Introduction Example	1021	2	1012	3
Clock System	8877	1	3390	2
Moving Button System	12	1	10	2

Table 2: Time consumption

Those results calculated using memory usage and time usage; result shows bugs found by developed prototype and netbeans debugger. While analysing results memory usage were divided by total found bugs for analysing total required memory for particular system and total time were divided by total found bugs for analysing time usage required for particular system.

V. CASE STUDY

We are here giving input program for debugging. As input taken for debugging from user the output generated by developed system. We can see here performance of system in terms of Heap memory and loaded class count also.



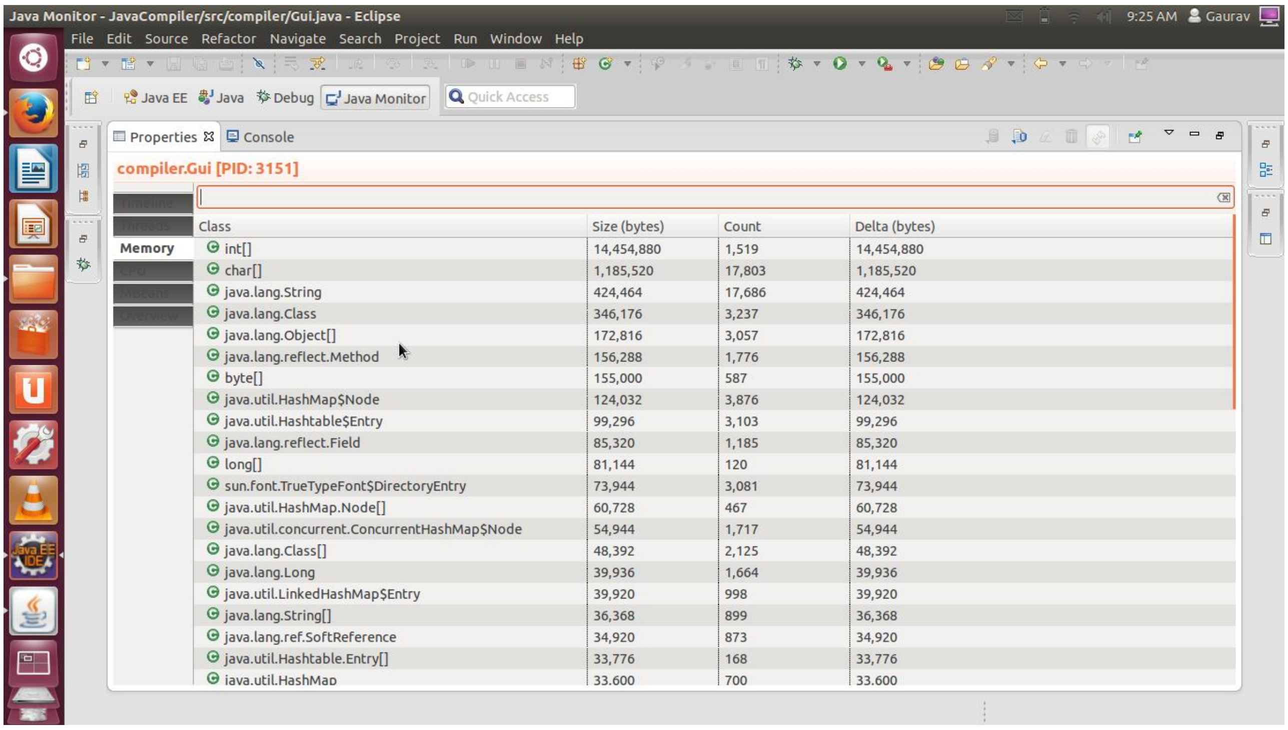
Screen 2 shows exception of process id in terms of size, count and bytes.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

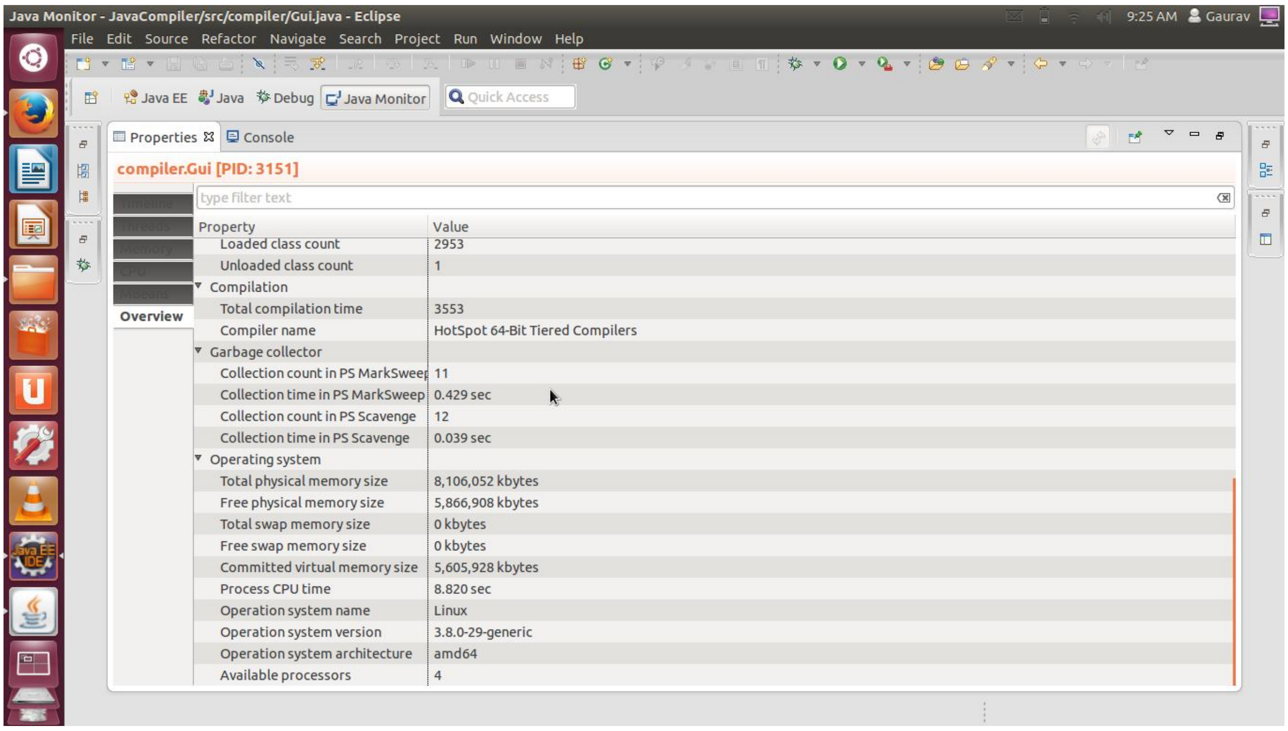
Vol. 3, Issue 7, July 2015


Java Monitor - JavaCompiler/src/compiler/Gui.java - Eclipse
9:25 AM
Gaurav
File Edit Source Refactor Navigate Search Project Run Window Help
Java EE
Java
Debug
Java Monitor
Quick Access
Properties
Console
Memory

compiler.Gui [PID: 3151]

Class	Size (bytes)	Count	Delta (bytes)
int[]	14,454,880	1,519	14,454,880
char[]	1,185,520	17,803	1,185,520
java.lang.String	424,464	17,686	424,464
java.lang.Class	346,176	3,237	346,176
java.lang.Object[]	172,816	3,057	172,816
java.lang.reflect.Method	156,288	1,776	156,288
byte[]	155,000	587	155,000
java.util.HashMap$Node	124,032	3,876	124,032
java.util.Hashtable$Entry	99,296	3,103	99,296
java.lang.reflect.Field	85,320	1,185	85,320
long[]	81,144	120	81,144
sun.font.TrueTypeFont$DirectoryEntry	73,944	3,081	73,944
java.util.HashMap.Node[]	60,728	467	60,728
java.util.concurrent.ConcurrentHashMap$Node	54,944	1,717	54,944
java.lang.Class[]	48,392	2,125	48,392
java.lang.Long	39,936	1,664	39,936
java.util.LinkedHashMap$Entry	39,920	998	39,920
java.lang.String[]	36,368	899	36,368
java.lang.ref.SoftReference	34,920	873	34,920
java.util.Hashtable.Entry[]	33,776	168	33,776
java.util.HashMap	33,600	700	33,600

This Screen shows different properties of debugging program which is not in traditional debugging tools.


Java Monitor - JavaCompiler/src/compiler/Gui.java - Eclipse
9:25 AM
Gaurav
File Edit Source Refactor Navigate Search Project Run Window Help
Java EE
Java
Debug
Java Monitor
Quick Access
Properties
Console
type filter text
Overview

compiler.Gui [PID: 3151]

Property	Value
Loaded class count	2953
Unloaded class count	1
▾ Compilation	
Total compilation time	3553
Compiler name	HotSpot 64-Bit Tiered Compilers
▾ Garbage collector	
Collection count in PS MarkSweep	11
Collection time in PS MarkSweep	0.429 sec
Collection count in PS Scavenge	12
Collection time in PS Scavenge	0.039 sec
▾ Operating system	
Total physical memory size	8,106,052 kbytes
Free physical memory size	5,866,908 kbytes
Total swap memory size	0 kbytes
Free swap memory size	0 kbytes
Committed virtual memory size	5,605,928 kbytes
Process CPU time	8.820 sec
Operation system name	Linux
Operation system version	3.8.0-29-generic
Operation system architecture	amd64
Available processors	4

VI. Conclusion and Future Work

Debugging is an important aspect in software development, where the ultimate target may be to develop code which is nil from any defect issues, Software engineers must live in the real and inadequate world where system complexity is essential thing will go to occur rapidly, Due to this The reality that debugging remains the most problematic and economical costly issue of the development cycle which indicates that engineers must work on the idea of day to day increasing complexity, prepare for the challenge from day one and acquire more as well as better debugging tools to keep increasing demands on developments. Programmers have been waiting a long time for practical automated debugging tools, they already gone a long way from near the beginning days of debugging to further advance the state of the art in this scenario, must push research towards more capable directions that take into account the way programmers actually debug in real scenarios. This proposed system provides a suitable support for the developer facilitating different features which will ultimately improve the performance of object oriented software's and their applications.

ACKNOWLEDGMENTS

It is my great pleasure to express my deep sense of gratitude to my project guide Prof. M.C. Kshirsagar (PG Coordinator), for his valuable guidance, inspiration and wholehearted involvement during every stage of this paper presentation.

I am very much grateful to the entire VACOE for giving me all facilities and work environment which enable me to complete my task. I express my sincere thanks to Prof. S. G. Joshi Head of Computer Department, VA College of Engineering, Ahmednagar who gave me their valuable and rich guidance and help in presentation of this research paper.

References

Adrian lienhard, tudor Girba and Oscar Nierstrasz "Practical Object Oriented Back-In-Time Debugging"LNCS 5142, pp 592-615.
[2] Raimondas Lencevicius, Urs Holzle And Ambuj K. Singh, "Query-based Debugging of Object-Oriented Programs" OOPSLA 97 Atlanta, USA.
[3]Mark Minas "Cyclic Debugging For pSather, a Parallel Object-Oriented Programming Language" Jan 31 2002
[4]Tanja Mayerhofer,"Testing and Debugging UML Models Based On Fuml" ICSE 2012.
[5]G. Pothier, E. Tanter, and J. Piquer, "Scalable omniscient Debugging, "Proceedings of the 22nd Annual SCM SIGPLAN Conference on Object Oriented Programming Systems, Languages And Applications (OOPSLA'07), vol. 42, no. 10, pp.535–552, 2007.
[6]C. Hofer,M. Denker, and S. Ducasse, "Design and implementation of a backward-in-time debugger," in Proceedings of NODE'06, ser. Lecture Notes in Informatics, vol. P-88 (GI), Sep 2006, pp. 17-32.
[7]J. KO and B. A. Myers, "Designing the whyline: a debugging interface for asking questions about program behaviour," in Proceedings of the 2004 onference on Human factors in computing systems. ACM Press, 2004, pp. 151–158.
[8] Sai Zhang; Zhongxian Gu; Yu Lin; Jianjun Zhao "AutoFlow: An automatic debugging tool for AspectJ software" ICSM 2008. IEEE International Conference on 2008, pp. 470 – 471.
[9] Rossler, J. "How helpful are automated debugging tools?" User Evaluation for Software Engineering Researchers (USER), 2012 IEEE Conference Publications, pp. 13 – 16.
[10] Chi-Neng Wen;shu-hsuan Chou;chih Chen ;tien-fu chen."NUDA: A Non-Uniform Debugging Architecture and Nonintrusive Race Detection For Many core system" IEEE transaction, vol.61, 2012, pages.199-212.
[11]Engblom, J. "A Review of Reverse debugging" System, Software, SC and Silicon Debug Conference (S4D), 2012, pp. 1 – 6.
[12]Chris parnin and alessandro orso, "Are automated debugging techniques actually helping programmers" ISSTA' July 2011
[13]Jorge ressia, Alexandre Bergel and Oscar Nierstrasz "object centric debugging" ICSE 2012
[14]Renee McCauley, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lynda Thomas and Carol Zander "Debugging: a review of the literature from an educational perspective" June 2008
[15]K. Glerum, K. Kinshumann, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle, and G. Hunt, "Debugging in the large: ten years of implementation and experience," *Proc. SOSP*, 2009, pp. 103-116.
[17]Noor Fazlida Mohd Sani, Noor Afiza Mohd Arifin and Rodziah Atan "Design of object-oriented debugger model using unified modeling language" JCSSP 2013, pp 15-18.
[18]Potanin, A., Noble, J., Biddle, R.: Snapshot query-based debugging. In: Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'04), Washington, DC, USA, IEEE Computer Society (2004) 251
[19]P. Iyenghar, C. Westerkamp, J. Wuebbelmann, E. Pulvermueller, A Model Based Approach for Debugging Embedded Systems in Real-time, in 10th ACM international conference on Embedded Software, EMSOFT 2010, ACM, New York, NY, USA, 69-78.
[20]Ahmadzadeh, M., Elliman, D., & Higgins, CNovice programmers: An analysis of patterns of debugging among novice computer science students. (2005). Inroads, 37(3), 84 88.
[21]Robertson, T., Prabhakararao, S., Burnett, M., Cook, C., Ruthruff, J., Beckwith, L., et al. (2004). Impact of interruption style on end-user debugging. In E. Dykstra Erickson & M. Tscheligi (Eds.), Proceedings of the 2004 conference on human factors in computing systems (pp. 287 294).
[22] D. M. Thakore & Tanveer S Beg "Develop object based debugging tool technique for improving the performance of object oriented softwares" , IJCSEITR, Vol. 3, Issue 5.