International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:8, No:9, 2014

A Novel Method for Live Debugging of Production Web Applications by Dynamic Resource Replacement

Khalid Al-Tahat, Khaled Zuhair Mahmoud, Ahmad Al-Mughrabi

Abstract—This paper proposes a novel methodology for enabling debugging and tracing of production web applications without affecting its normal flow and functionality. This method of debugging enables developers and maintenance engineers to replace a set of existing resources such as images, server side scripts, cascading style sheets with another set of resources per web session. The new resources will only be active in the debug session and other sessions will not be affected. This methodology will help developers in tracing defects, especially those that appear only in production environments and in exploring the behaviour of the system. A realization of the proposed methodology has been implemented in Java.

Keywords—Live debugging, web application, web resources, inconsistent bugs, tracing.

I. Introduction

IT is inherent in the nature of software development to have undetected bugs that pass into the production environment. A software bug is defined as an occurrence of a system design or program source code that does not meet a requirement [5]. The purpose of debugging is to locate the offending piece of code and to execute and perform necessary modifications to fix it [6]. In fact, even software testing and verification cannot guarantee the absence of bugs, but they can guarantee their presence [1].

Some bugs are a result of implementations based on incomplete or inaccurate specifications [2]. Debugging is an important and continuous activity in the maintenance phase of the software life cycle. Knowing that software maintenance costs about 90% of the overall system costs [3] and that the testing and debugging activities constitute from 50% to 70% of development costs [4], it is important to have debugging methodologies and techniques that aid in the quick detection and fixing of bugs.

Debugging occurs during three stages of the software development; the actual development and coding of the system, software testing, and finally when the system is on production. During development, the developer may easily perform any code changes necessary to debug. At this stage, the developer usually runs an instance of the system on his development machine, and there is no cost associated with performing changes. During the testing phase, the system is usually deployed on machines dedicated for the quality control team to perform testing and verification.

Khaled Al-Tahat(ktahat@yahoo.com) is associated with Open Arab University and Khaled Mahmoud(Khaledinho@yahoo.com) is associated with Middle East University. Ahmad Al-Mughrabi(ahmad.al mughrabi@live.com) is associated with Zarqa University

Performing code changes and deploying them on test and production servers for the purpose of debugging is alluring because it enables developers to inspect the effects of these changes and gain better understanding of the behaviour. This is particularly useful in case of inconsistent bugs that appear on one environment (production environment for example) only. However, deploying these changes, usually, interrupt the functionality of the application because a server restart is required. Moreover, these changes whose effects should only be noticed by developers become incorporated into the system affecting all users.

In this paper, we propose a methodology that enables software developers and maintenance engineers to replace web resources on the production server so that these resources are only active in their web sessions.

Section(2) of this papers presents the literature and related work in the field of debugging and compares our methodology to the existing methodologies.Section(3) provides the necessary background related to request processing and introduces the concept of web resources. Section(4) presents the model of the proposed methodology A simple implementation is presented in section(5). Finally, section(6) discusses the limitations and possible future enhancements.

II. Literature Review and Related Work

Software debugging has been heavily researched in the literature and many approaches and solutions have been developed and investigated such as static slicing, dynamic slicing, automated debugging and relative debugging. In this section we provide a summary of some of these works and state the position of our proposed methodology with respect to the literature.

Program slicing research is concerned with building algorithms that locate all possible execution paths to a specific statement in a program and in harnessing these algorithms for bug fixing and re-factoring. Slicing algorithms accept a criterion specifying a set of variables at the target statement as their input. When the criterion is passed, all possible execution paths that lead to that statement and that affect the values of the variables at the criterion are included in the resultant execution paths. Program slicing concept was first introduced by Wiser [7]–[10]. Subsequent researches focused on methods and techniques of finding program slices and on utilizing program slices to quickly and efficiently solve

software bugs. The notion of program dependence graphs [11], [12] was introduced as a structure to be adopted by the slicing algorithms.

Program slicing techniques are also utilized by other application domains. Examples of these applications include component identification [13] and elimination of unnecessary loops in the source code re-factoring domain [14].

DARWIN [15] is a promising methodology that utilizes dynamic slicing to support automatic discovery of regression bugs. A regression bug is introduced due to changes made in the source code and it that did not exist in the previous version before the changes had been made. This methodology, captures two dynamic slices for a specific input that succeeds in the original program and another slice for an input that follows the same path of a successful input at the original program but a different path in the buggy program that fails. Those two slices are compared and possible culprit code locations are pointed out.

RES [16] aids in debugging by synthesizing an execution suffix leading to an input core dump for an input program. Inspecting the execution suffix for a faulty execution in a backward fashion helps developers to detect the root cause quickly.

Relative Debugging [17] is a set of tools that enable developers to issue debugging commands to compare states (variables for example) of two versions of the same program running at the same time. This aids in detecting bugs specially if the defect is present on one version (production) but is not present on the other. This research included developing artefacts for Eclipse and .NET.

JRebel [18], [19] is a deployment tool that enables developers to modify source code and deploy it to a web server without the need to restart the server and thus interrupt the service. Although it is not directly related to debugging, JRebel is included the literature because it employs the concept of live hot deployment of compiled Java source files.

In our proposed methodology, software maintenance engineers deploy resources at the production server without the need to restart the server and those resources are only active in a specific session and upon request. Compared to other methodologies, our methodology focuses on the production environment where the bug was discovered by enabling software maintenance engineers to perform changes and inspect the effects of those changes on the flow of the web application. Our methodology is distinguished by inflicting no side effects on the production environment since the resources will be only active for a specific session.

III. Request Processing and Web Resources

Before presenting the live debugging proposed methodology, it is important to have basic understanding of how web servers respond to requests. A traditional web server may host more than one web application with each being distinguished by its IP address, host name or context path. In case of having more than one application that share the same IP address or host name, an application is distinguished by its context path. Consider the following

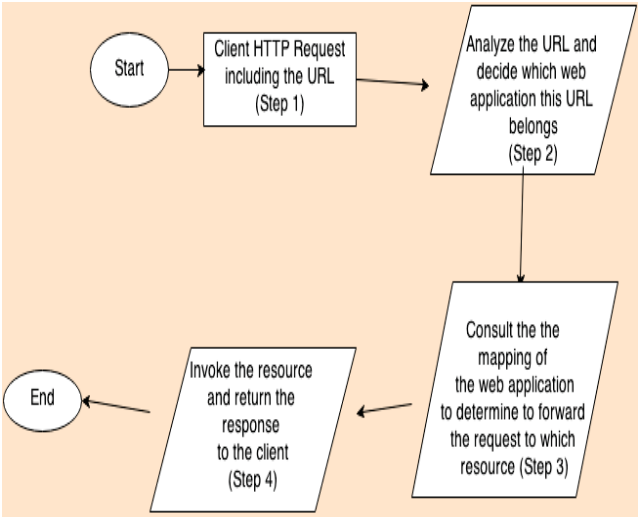

Fig. 1. Request Processing Life Cycle

example URLs ”127.0.1.1/Application1/index.jsp” and ”127.0.0.1/Application2/index.html”. These URLs are for two applications deployed at the same server. The context path for the first URL is ”Application1” where the context path for the second URL is”Application2”. The requested resource path for the first URL is ”/index.jsp” and the requested resource path for the second URL is ”/index.html”.

A web resource is either static or dynamic. Examples of static resources include images, HTML files, CSS files and JavaScript files. Dynamic resources represent programs (source code) that are executed when requested and they dynamically build responses based on parameters and inputs from the request. Examples of dynamic resources include: JSP(Java Service Pages), PHP, and Java Servlets.

A web application is a collection of dynamic and static resources and a map between URL patterns and these resources. When a request is received, the web server inspects this map to decide which resource to invoke when a request is issued. This map is built automatically upon the deployment of the web application by the hosting web server.

Fig.1 illustrates the request processing lifecycle. When a request is received(Step 1), the server determines the target web application of the request (Step 2).Then it inspects the resource mapping of the web application to determine which resource to invoke(Step 3). Finally, the resource is invoked (Step 4) and the proposed methodology operates right before invoking the resource to determine if another debugging version of the resource exists and invoke it instead of the requested resource.

IV. The Live Debugging Proposed Methodology

The proposed live debugging methodology enables software maintenance engineers to deploy another version of an existing resource for the purpose of debugging. The debugging version may contain extra code to better help understand a use case or debug a defect. Examples of these changes include, printing the values of some variables on the screen, changing specific

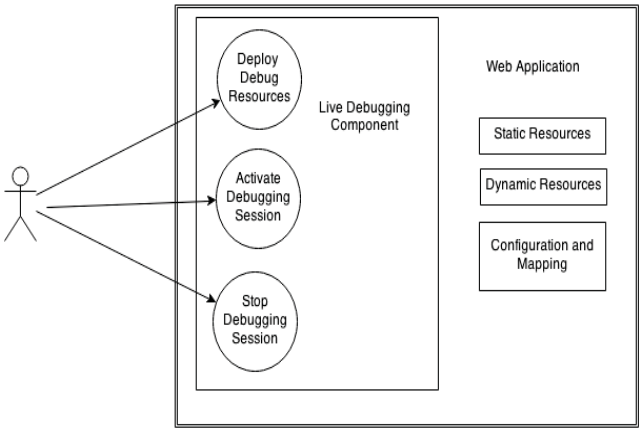

Fig. 2. Request Processing Life Cycle

variable values to inspect their effect on the output. The debugging version of the resource is activated by issuing specific commands to the web application and will only be available to the session that issued the activation commands. Actions that a software maintenance engineer can perform using the proposed methodology are shown in Fig.2.

As Fig.2 illustrates, the live debugging component is included as a module inside the web application. It provides extra functionality to enable the developer to deploy the resources, start an debugging session, and stop the active debugging session. Those actions are performed through a graphical user interface.

Live debugging has been realized by intercepting the invocation of web resources (Step 4 of the request processing life cycle shown in Fig.2).Fig.3 summarizes the steps involved in invoking the debugging version of a resource. After intercepting a request, the interceptor determines whether the current session is a debugging session (Step 2). If it is a debugging session, the interceptor determines whether there is a debugging version deployed and invokes it if it exists. Otherwise, the original version is invoked.

In the proposed methodology, software maintenance engineers deploy a group of resources as a change set. A change set has a unique name per web application and includes the new versions of the resources (new resource files) .

Table I presents an example of a change set. When this change set is deployed into the server, the new resource files will be deployed at the locations dictated by the change set. The interceptor inspects this mapping and determines that the path for the resource ”/cart/pay/jsp” should be ”/cart/debug/pay.jsp” for the current session. If there is no entry for a given resource in the change set, its original version will always be invoked.

TABLE I
EXAMPLES OF RESOURCES DEBUGGED BY A CHANGE SET

Resource Name (Path)	New Resource File Paths
/cart/pay.jsp	/cart/debug/pay.jsp
/home/preferences.jsp	/home/debug/preferences.jsp

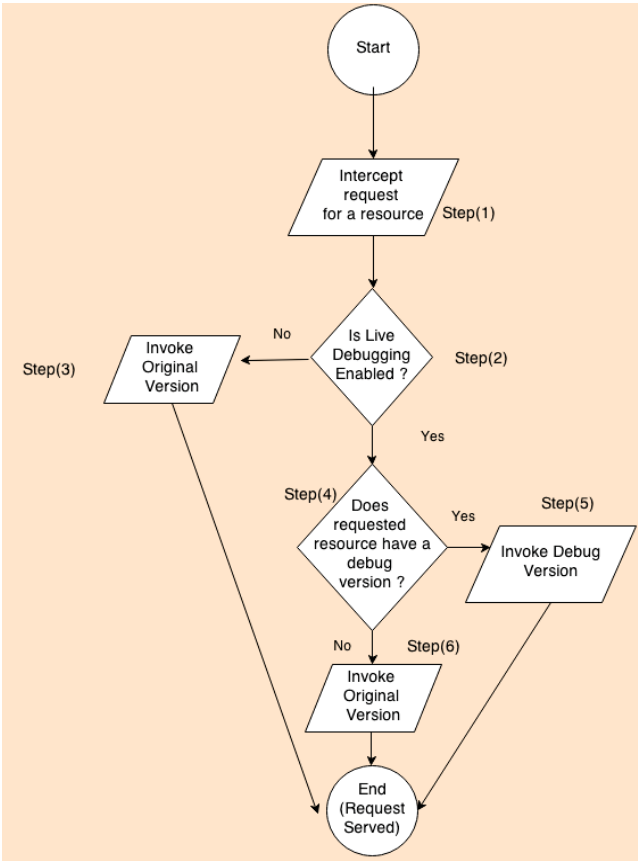

Fig. 3. Invoking a debugging version of a resource

Most web application frameworks and development languages enable developers to intercept the normal request processing life cycle in order enabling them to perform custom tasks (Step 4 of Fig.1).Examples of these tasks include decryption and applying intrusion detection algorithms. In the live debugging methodology, the request is intercepted to invoke the new version of a resource if it exists.

Fig.4 further illustrates the interception process. The ”intercept” method has access to the data session of the corresponding HTTP request. It determines whether the current session is a debugging session by inspecting an attribute in the server http data session (Line 3). If this attribute is true, the active change set is loaded (Lines 4-5). Then, the interceptor determines if the requested resource path is included in the change set (Line 7). If it is included, then the resource located at the new path is invoked (Lines 7-8). Otherwise, the original resource is invoked (Line 9).

V. SAMPLE IMPLEMENTATION

We implemented our methodology in Java. The interceptor mechanism in Java enables developers to intercept invocation of web requests[20] by enabling them to provide their own interceptor classes as part of the web application. These interceptors are called ”filters” in Java web application

```
1 Inputs url: URL of the requested resource , session: The session data of the current user
2 intercept_request(requested_resource_path, session){
3 DEFINE do_debug = is_debug_session(session)
4 IF do_debug = TRUE THEN
5 DEFINE active_change_set = get_debug_change_set(session)
6 DEFINE path_included = does_changeset_include_resource(active_change_set, requested_resource_path)
7 IF path_included = TRUE THEN
8 invoke(new_path)
9 ELSE invoke(request_resource_path)
10 END IF
11 ELSE invoke(requested_resource_url END IF}
```

Fig. 4. Intercept Source Code

terminology. More than one filter for a given resource may exist. Before the Java web container invokes a resource, it determines if filters exist. If so, it delegates the invocation of the resource to those filters according to the order of their configuration. When a web filter completes its task (decryption for example), it delegates the invocation of the resource to the next filter. When all filters complete their work, the web container invokes the resource.

At any time, a filter may decide to stop the invocation of a resource and forward to the request to another resource on the server. This is performed through the Java request dispatching API (javax.servlet.RequestDispatcher).

Fig.5 is an implementation of the abstract functionality described by Fig.4 .The web container passes a request object to the filter containing all necessary information about the HTTP request(Lines 1-3). The filter reads the URL of the requested resource (Line 8) and checks if the current session is a debugging session (Line 11) and if it is a debugging session and the resource has a debugging version Line(14) it forwards the request to that debugging version (Line 22-23). It important to notice here, that a RequestDispatcher is used to forward the request by passing it the URL of the new resource to invoke. In case of no debugging, the request proceeds normally delegating to the next filters the task of the resource invocation (Line 20).

After creating the resource filter class, the web developer has to tell the web container what resources to intercept and delegate to the filter. This is performed through the web application configure file (web.xml).

Our implementation enables replacing the following types of static resources : cascading style sheet files, various image files (png, jpeg, bmp, gif) and Javascript files. The implementation also supports replacing one type of dynamic resources :script files (JSP-s).

VI. Conclusion and Future Work

In this work, we have proposed an innovative method for live debugging. This method saves developers efforts and time while tracing bugs. It can also be used as a learning

```
1 void doFilter(ServletRequest request,
2 ServletResponse response,
3 FilterChain chain)
4 throws IOException,
5 ServletException
6 HttpServletRequesthttpReq = (HttpServletRequest) request;
7 HttpSession session = httpReq.getSession();
8 String resourcePath = httpReq.getServletPath();
9 booleanproceedNormally = true;
10 String newResourcePath = null;
11 if (session.getAttribute(Settings.ACTIVE_SET) != null) {
12 Map changeSet = (Map) session.getAttribute (Settings.SESSION_ATTR_ACTIVE_SET);
14 if (changeSet.containsKey(resourcePath)) {
15 proceedNormally = false;
16 newResourcePath = changeSet.get(resourcePath);
17 }
18 }
19 if (proceedNormally) {
20 chain.doFilter(request, response);
21 } else {
22 RequestDispatcherreqDispatcher = request.getRequestDispatcher(newResourcePath);
23 reqDispatcher.forward(request, response);
24 }
```

Fig. 5. The Resource Debug Filter Implementation

tool since it enables developers to perform changes and inspect the effects of these changes without affecting the original functionality. We strongly suggest that companies like Microsoft, Oracle consider incorporating live debugging features to be inherent in their languages tools.

Future enhancements include research to replace compiled classes at runtime per request. This will enable developers to replace any Servlet classes and their dependencies which is not supported by our methodology and implementation.

References

[1] E. W. Dijkstra, *Notes on Structured Programming*,Editors, Academic Press, London, pp. 1-82,1972
[2] Ehud Y. Shapiro, *Algorithmic Program Debugging*, MIT Press, Cam-bridge, MA, 1983.
[3] B.Hailpern, P. Santhanam, *Software debugging, testing, and verification*,IBM Systems Journal, 2002, Vol.41, Issue.1.
[4] Lewish, G. Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices. Addison-Wesley, Boston(2003).
[5] B. Hailpern, P. Santhanam, *IEEE Guide to the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software*, IEEE Standard 982.2-1988, IEEE, New York (1989).
[6] R.McCauleya, .Fitzgeraldb, G.Lewandowskic, L.Murphyd, B.Simone, L.Thomasf and C.Zanderg , emphDebugging- A Review of the literature from an educational perspective, Computer Science Education, pages 67-92,Volume 18, No. 2,June 2008
[7] F.Tip, emphA Survey of Program Slicing Techniques,Technical Report, CWI, Amsterdam, The Netherlands, 1994.
[8] M. Weiser, emphProgram slices: formal, psychological, and practical investigations of an automatic program abstraction method, PhD thesis, University of Michigan, Ann Arbor, 1979.

[9] M. Weiser, emphProgrammers use slices when debugging, Communications of the ACM, pages 446-452 Volume 5, Issue 7 1982.
[10] 10- M. Weiser, *Program slicing*, IEEE Transactions on Software Engineering, 352357, Volume 10, Issue 4 ,1984.
[11] J. Ferrante, K.J. Ottenstein, and J.D. Warren, *The program dependence graph and its use in optimization*, ACM Transactions on Programming Languages and Systems, pages 319-349, Volume 9, Issue , 1987.
[12] F. Bergeretti and B.A. Carre, *Information-flow and data-flow analysis of while-programs*, ACM Transactions on Programming Languages and Systems, 3761, Volume 7, Issue 1 ,1985.
[13] N.Rodrigues, emphComponent Identification Through Program Slicing, Electronic Notes in Theoretical Computer Science, pages 291304 Volume 160, 8 August 2006.
[14] T.Amtoft, *Slicing for modern program structures: a theory for eliminating irrelevant loops*, Information Processing Letters, Pages 45-51, Volume 106, Issue 2, 15 April 2008.
[15] D.Qi, A.Roychoudhury, Z.Liang, and K.Vaswani, *DARWIN: An approach to debugging evolving programs*,ACM Transactions on Software Methodolgy Methodology, Volume 21, Issue 3 ,July 2012
[16] C.Zamfir, B.Kasikci, J.Kinder, E.Bugnion, G.Candea, *Automated debugging for arbitrarily long executions*, In Proceedings of the 14th USENIX conference on Hot Topics in Operating Systems (HotOS'13). USENIX Association, Berkeley, CA, USA,Year:2013.
[17] D.Abramson, C.Chu, D.Kurniawan1, A.Searle, emphRelative debugging in integrated development environment, SoftwarePractice and Experience , pages1157–1183. Volume 39, Issue 14, Year 2009.
[18] J.Kabanov, *JRebel Tool Demo*, Electronic Notes in Theoretical Computer Science Volume 264 Issue.14, Year 2011.
[19] JRebel Website, zeroturnaround.com/software/jrebel, 5/5/2014
[20] Servlet Filters, http://www.oracle.com/technetwork/java/filters-137243.html, 5/5/2014

Khalid Al-Tahat Khalid Al-Tahat holds a BSc in Computer Science from Yarmouk University in Jordan, an MSc. in AI from the Malaysian University for Science and Technology and a PhD in Software Engineering from the National University of Malaysia. Dr. Al-Tahat worked at different educational institutes in UK, Malaysia and Jordan. His research interest lies in the fields of modern Software Engineering, AI and Teaching Computing. He is author of about 20 papers published in international journals, conference proceedings, and invited book chapters.

Khaled Zuhair Mahmoud Khaled Mahmoud is a senior software engineer with more than six years of experience in Software Development. Khaled Mahmoud worked at various companies which allowed him to gain diverse technical and business knowledge. Technical Expertise includes developing core applications including multi-threaded applications, web applications, Enterprise Java Beans, JMS, CSharp, ASP.NET, Oracle PL/SQL, Grails, Oracle ADF and JSF. Khaled Mahmoud holds a Bachelors degree in Software Engineering with an excellent rating from the Hashemite university and a Masters Degree in Computer Science with excellent rating from the Middle East University.

Ahmad Al-Mughrabi Ahmad Al-Mughrabi holds a Bachelors degree in Computer Science from Zarqa University with excellent rating. His research interests include Software Engineering and Image processing. His great passion for Software Engineering along with his technical skills and expertise enable him to participate and contribute to scientific research and detect areas where scientific research can provide an added value science.