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

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

Profiling REST Services with Advent of Aspect Oriented Modelling

Sohel S. Shaikh, Dr. V. K. Pachghare

M. Tech. Student, Dept. of Computer Engineering, College of Engineering, Pune, India

Associate Professor, Dept. of Computer Engineering & I.T., College of Engineering, Pune, India

ABSTRACT: REST, a communication protocol is used for communicating via APIs. It is widely used today for intercommunication among the services and the users. However, it does have the constraint of constructing servers to be stateless. Stateless servers are restricted from storing any information about the clients. Hence profiling such resources becomes a serious issue, as session is not retained. AOP enables implementing cross cutting concerns to be applied to applications without disturbing their business logic. This paper aims to overcome this challenge of resources profiling using REST and AOP concepts. It presents a plug and play approach for profiling REST resources using Aspect Oriented Programming.

KEYWORDS: Aspect Oriented Programming, RESTFul web services Profiler, Aspect-Orientation Modeling, Crosscutting behavior, Profiling.

I. INTRODUCTION

REST- REpresentational State Transfer is an ability of applications to exchange and consume desired information. REST obliging services permit imploring services to access and handle literal representations of resources using an even set of operations.

Crafting a "RESTful" API does not simply imply placing an HTTP cover around an existing innate API. Nonetheless, REST is not a standard, it is an architectural grace. There are several customs to deduce this architectural style. Even though REST is not coupled to HTTP, commonly it is delivered lying on HTTP.

Furthermore, it is critical to leverage a common organizational structure, skills, processes and practices that are focused on designing, developing, testing and deploying enterprise-wide, robust APIs. This paper focuses on providing an alternative to maintain sessions using Rest services and AOP – Aspect Oriented Programming. It provides an innovative solution for profiling the rest resources. In certain applications, it is required to keep a track of users or services that access the resources via the provided APIs. This can be well achieved by using the concepts provided by the Aspect Oriented Software Development.

This paper presents a novel plug and play technique to achieve a modular solution. The profiling shall be formulated using the event streams to publish event records in the event log files. This approach is well elaborated in further sections.

The paper is designed as: Section 2 contributes an extensive outline of the concepts that influenced the proposed solution. Section 3 builds the background required to be familiar with REST and AOP concepts. It also provides the statelessness issue that is to be handled. Section 4 presents the methodology incorporated for developing the proposed solution. Section 5 offers the mathematical model. Section 6 gives the implementation details. Section 7 gives the Experimental results. Section 8 provides the outcomes and knowledge derived after implementing the solution.

II. LITERATURE SURVEY

The paper by Fielding, Roy Thomas describes the software production ideologies administering REST and the collaboration restraints selected to preserve principles, opposing them to the restrictions of other architectural flairs. It also provides a detail results and explanations of putting on REST to the policy of the HTTP and URI standards, and from their consequent disposition in Web consumer and producer software. [1] Roughly few papers have explored architectural styles for secure communications using REST. However, they do not comprise sessions clearly. Single

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

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

sign-on and delegation of authorities also lie exterior of their space. The paper [5] provides a brief theory on the challenges that are identified on using of REST APIs. The paper provides concepts and methodologies of Aspect oriented programming. [13] All these available theory and experimental results gave this paper inspiration to develop the proposed solution for profiling of REST resources.

III. THEORETICAL BACKGROUND

A. *REST*

Rest was first described in Roy Fielding's Research – 2000. It has the following six constraints as depicted in Figure 1.

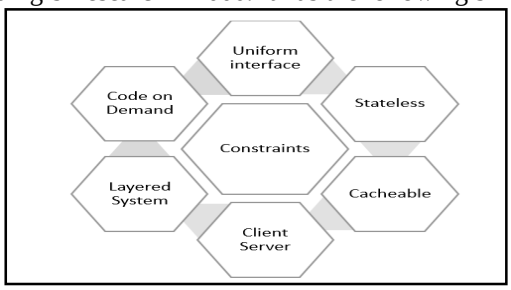

Figure 1 REST Constraints

Components of REST are as follows:
1. Resource
2. Resource Locator/Identifier (URIs)
3. Resource Representation (JSON)
4. HATEOAS (Connectedness): Resources are connected to each other using links
5. Communication Protocol - HTTP
a. Standard Methods
GET, POST, PUT, DELETE, HEAD, PATCH
b. Standard Responses

The overview of REST Services is illustrated in Figure 2.

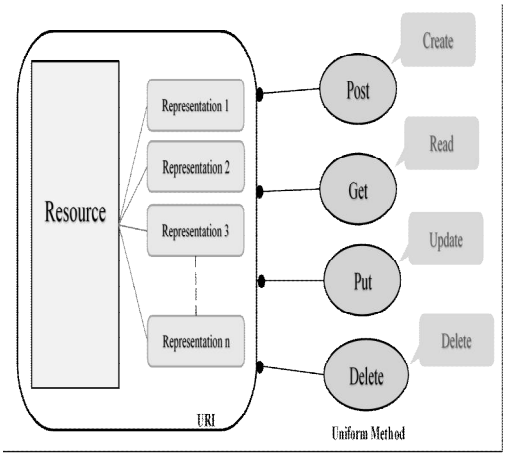

Figure 2 REST Services

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

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

B. *Aspect Oriented Paradigm:*

AOP is a complement to the traditional object oriented programming rather than being an alternative. It provides solutions to the various cross cutting concerns in the application that required redundant handling and lines of code for its execution. It makes the system simpler by separating these concerns into aspects. The Figure 3 portrays the AOP weaving concept. The legacy systems highly benefit from these features as the upcoming improvements and modifications now don't require any fidelity with the legacy code but instead just require aspects to code it separately and work efficiently. Figure 4 shows the building blocks of AOP.

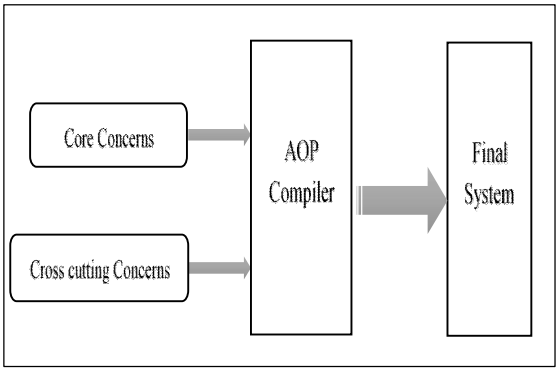

Figure 3 AOP weaving

Another chief region of alarm is code tangling. It is triggered when a unit is employed to grip various concerns at the same time. In multi-dimensional domain, the concerns that are implemented are independent and isolated from each other. Yet they need to be coded together. As the implementation domain in one dimensional in space. AOP here comes to the rescue. It lets the developer focus on the primary business logic only in the code, and enables them to handle various other cross cutting concerns using Aspects. This dilutes their confusion and allows them to enforce strong business logic of the application.

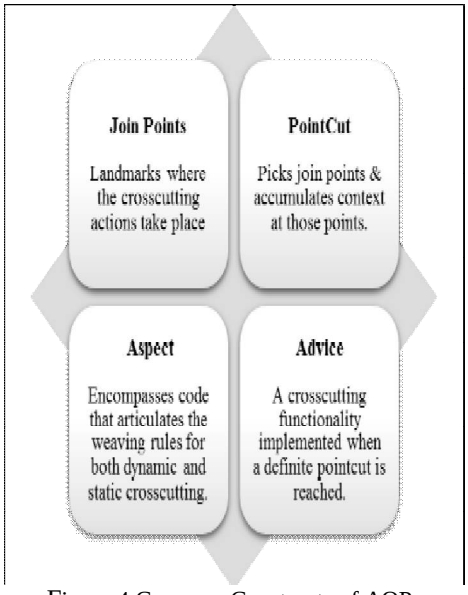

Figure 4 Common Constructs of AOP

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

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

AOP offers three kinds of advice:

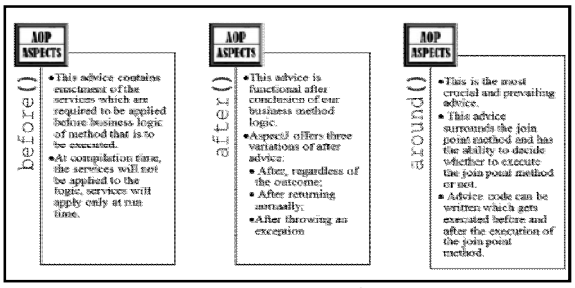

Figure 5 AOP Types of Advice

Code injection permits alteration of compiled code units to which the source codes are not accessible. The imposed code can be used to collect several run-time statistics in background where the traditional tools viz. profilers and debuggers cannot be used. The code can be imposed either at compile time, or when the compiled units being loaded by the runtime environment. By means of code injection, modifications of the original source files are not required, so it is fruitful in scenarios where injected codes need to be modularized, needs to be easily pluggable.

Profiling of REST resources is usually done with the intermediary components (i.e., proxies and gateways). These gears are commonly appended to an architecture accommodating the REST style to accomplish access control, caching, etc. Conversely, authentication and delegation technologies that count on session state depart from the REST flair.

C. *Statelessness disputes*

The statelessness REST constriction implies that servers should be stateless and should not sustain any conversation state with the client. Conversely, the isolation proposition states that any intermediary transformation of a transaction should not be detectible to ongoing parallel transactions. This obliges servers to preserve transitional states for actions that are not committed by sustaining a session state for a transaction. Consequently, these two properties are in conflict.

A session management in REST is not desired owing to the statelessness property. The system must be capable to validate authorization or authentication information. The shortcoming of performing authentication on every request is that one needs to lookup the login and password each time. Hence a token a used after first authentication request. The later conversation involves just exchanging of these tokens for authentication.

This paper proposes a tangential technique to implement these trepidations. Business logic is not disturbed much by this approach. Only few annotations and aspects are requisite to be pragmatic.

IV. METHODOLOGY

This paper proposes an Aspect Oriented model for profiling the REST resources over the fly. The profile is maintained without interrupting the tasks of the user. Aspects make it possible to append the additional code for profiling without altering the business logic. Figure 6 gives a pictorial view of the proposed model.

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

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

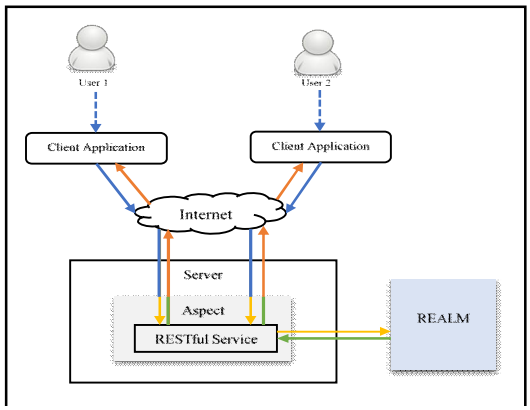

Figure 6 Block Diagram

The proposed methodology to profile REST resources is as follows:

1. Capture SecurityContext object.
   SecurityRealm is used to authenticate and authorize the users of the application. SecurityContext object is captured when user is authenticated and authorized by SecurityRealm to access REST endpoint. Which in turns provides authorized user details accessing REST API
2. Identify JoinPoint:
   JoinPoint are units where the aspect logic must be executed. It is crucial to identify our concern JoinPoint in the application. The JoinPoint of interest is the call to our REST API.
3. Implement PointCut
   Apply a PointCut around the identified JoinPoint.
4. Advice this PointCut
   Implement an advice in the aspect to be implemented at the PointCut. This advice is responsible for profiling the invoked REST resource.

V. MATHEMATICAL MODEL

Input : User requests access REST resource
Output : REST Request-Response is profiled to Log and response returned to user

S= {s, e, A, B, User, Communication_channel, ResourceURI, SecurityRealm, securityContext, ProfilingAspect, Logs, RESTResource, $F_{request}$, Fauthenticate, F_{store}, F_{fetch}, $F_{forward}$, $F_{response}$, F_{store} …}

where,
S: Sample space
s: starting point of the implementation
e: ending point of the implementation
A: REST request from the client
B: REST response to client

In the beginning, a user makes a request for the REST resource using the ResourceURI

s: User→($F_{request}$(A)) Communication_channel(ResourceURI)

The SecurityRealm authenticates & authorizes the user

SecurityRealm→Fauthenticate(User)

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

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

On successful Authentication & Authorization the SecurityRealm stores the Users ID in securityContext
SecurityRealm →Fstore(User_ID)

After this, the request is intercepted by the Profiling Aspect
ProfilingAspect(r1) →Fcapture(RESTrequest)

The ProfilingAspect also fetches the User_ID from the securityContext
ProfilingAspect(s1) →Ffetch(securityContext(User_ID))

Then the ProfilingAspect concatenates both and stores it into the logs
Logs →Fstore (r1 + s1 + currentTimestamp)

The request is finally forwarded to the REST resource
ProfilingAspect →Fforward(A)

The REST resource is accessed and the response, say B is returned as response
REST Resource →Fresponse(B)

This response is again intercepted by the ProfilingAspect
ProfilingAspect(r2) →Fcapture(RESTresponse)

The ProfilingAspect again fetches the User_ID from the securityContext
ProfilingAspect(s2) →Ffetch(securityContext(User_ID)

Then the ProfilingAspect concatenates both and stores it into the log thus profiling the REST request
Log →Fstore (r2 + s2 + currentTimestamp)

Finally, the response is returned to the user
CommunicationChannel →Fresponse(B)

End of the program (e):

Success : Log is maintained with the userID, resource accessed and the timestamp at which the request & response takes place.

Failure : The user is not a legitimate user.

VI. IMPLEMENTATION

The paper presents a self-illustrative implementation plan using the figure (7). It comprises of the following steps. In the first step, a resource is created. These resources are consumed by the users of our application. These resources require secure access. Hence a log is to be maintained to examine the users accessing the resource. In the second step, REST endpoints were developed to communicate with these resources. These REST services are by nature desired to be stateless. Hence no sessions can be created to maintain the time span for which the users interacted with the available resources. Hence in the third step, the use of AOP concepts were employed. AOP – Aspect Oriented Programming concepts were used to serve the cross-cutting concerns of the developed application.

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

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

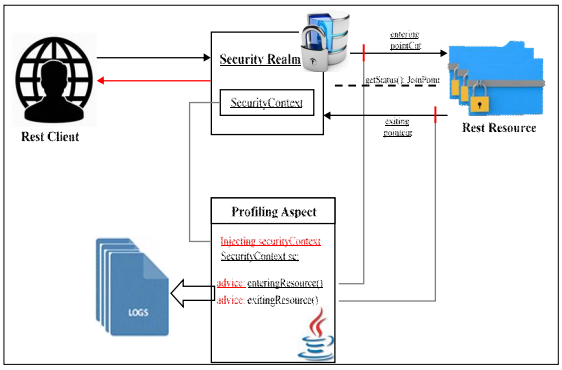

Figure 7 Working model

Whenever a user tries to access the resources, it does so using an REST API. These requests are trapped for yielding the users profile. The system authenticates the user using the security realm that is available. If the user withstands this check by matching its credentials, it can access the requested resource. This is the PointCut of our concern. Once the PointCut is triggered, an advice is executed from the aspect. This advice logs the User profile that is accessing the resource. When the request gets fulfilled again the advice is executed to log users' profile. This enables us to profile the REST resources.

This dodging of requests on the fly, is achieved using the novel technique of AOP PointCut. The REST call is wrapped with the AOP aspect. This aspect contains the around() advice. Advice: around() encompasses the join point method and can decide whether to execute the join point method or not. The developed advice code is executed before and after the execution of the join point method i.e. REST call. On execution of the advice, the user profile gets logged into the event stream for further profiling.

In this way, secured access to the resources accessed by the REST APIs is achieved. The users accessing the resources are logged for future investigations if required. Hence the need to maintain sessions violating the REST principle of statelessness is completely avoided.

VII. EXPERIMENTAL RESULTS

To gain an insight about the changes on application programs the implementation was evaluated against the open source tool provided by Apache Tomcat i.e. JMeter. It is a very efficient for evaluating Java Applications.

This paper implements the proposed solution using Java 8. It implements all the details mentioned in the above sections. To provide strong evidence related to the performance of the proposed solution the same REST Service was implemented without profiling using Aspect Oriented Modeling. The tool used to evaluate the performance measurement was JMeter.

Numerous tests were repeatedly executed in a loop to create reasonable stress on the application. 10 user threads were simulated each performing 1000 requests.

There were three functions that were executed:

(1) Enable Tasks

(2) Get Tasks

(3) Disable Tasks.

Both applications were tested against the same samples.

The difference between using REST APIs with and without profiling was examined as part of research. The results thus obtained after testing the two modules are as displayed below:

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

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

Label		Without Profiling	With Profiling
Enable Task	Samples	1000	1000
	Average	1	2
	Min	0	1
	Max	313	221
	Std. Dev	3.27	3.42
	Error %	0.00%	0.00%
	Throughput	799.9	668.4
	Received KB/sec	150.92	126.10
	Sent KB/sec	190.61	159.27
	Avg. Bytes	193.2	193.2
Get Task	Samples	1000	1000
	Average	0	0
	Min	0	0
	Max	10	8
	Std. Dev	0.60	0.68
	Error %	100.00%	100.00%
	Throughput	800.3	668.6
	Received KB/sec	85.33	71.29
	Sent KB/sec	187.56	156.7
	Avg. Bytes	109.2	109.2
Disable Task	Samples	1000	1000
	Average	1	2
	Min	0	1
	Max	360	221
	Std. Dev	3.67	2.63
	Error %	0.00%	0.00%
	Throughput	800.3	668.6
	Received KB/sec	150.98	126.14
	Sent KB/sec	191.47	159.96

Table 1 Comparison of With and Without Profiling Code

As seen in the above table there is not much significant change in the values measured for both. Also, in the below graphs the throughput is approximately, 143,976.964 / minute. This java module was implemented without the profiling aspects for REST APIs.

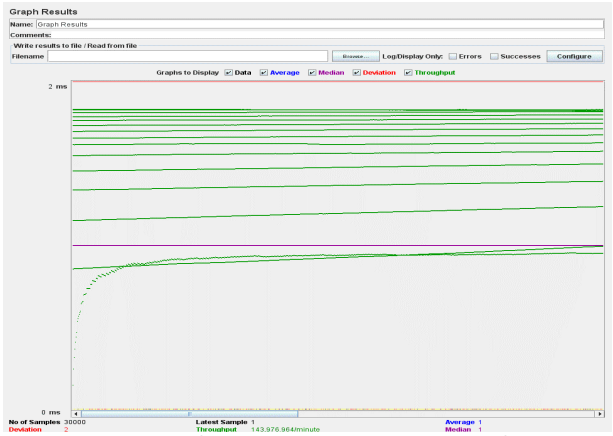

Figure 8 Graph for REST Services Without Profiling

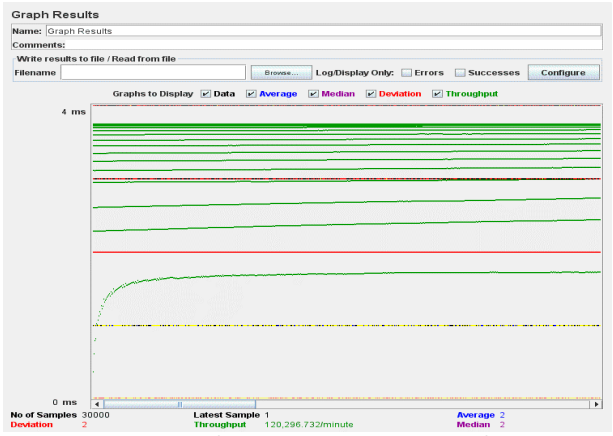

Figure 9 Graph for REST Services with Profiling

The above graph was generated for the java code that was implemented with the proposed Profiling methodology using Aspect Oriented Modeling. The throughput thus obtained from this code is 120,296.732 / minute. As seen from above two graphs the throughput thus decrease with the advent of profiling logic. However, it must be noticed that this trivial degradation in throughput is acceptable as it comes at the expense of an added functionality in the system.

Thus, it proves that the proposed profiling model is best for use with legacy systems as well. Where identified, crosscutting concerns can be advised for performing Profiling without any disturbance to the core business logic and yet not affect the systems performance.

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

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

VIII. CONCLUSION

This paper presents a novel technique of profiling REST resources. It eliminates the need for creating and maintaining sessions. Thus, RESTfulness is maintained as session is not getting tracked in REST procedures. Also, a Single point of control for profiling all the rest APIs is achieved. And hence, any future advancements can be reflected easily with minimal changes. Moreover, as a single @Around advice will be sufficient to develop a profile of all the REST APIs, Lines of code is minimal and logic is easy to understand. Thus, improving the understandability and maintainability of the code. This paper proposes a modular solution. It is a plug and play approach. Hence can be unplugged easily as per our requirements also it can be upgraded easily. Thus, it enforces a single point of control.

REFERENCES

1. Fielding, Roy Thomas. Architectural styles and the design of network-based software architectures. Diss. University of California, Irvine, 2000.
2. Pautasso, Cesare. "On composing RESTful services." Dagstuhl Seminar Proceedings. Schloss Dagstuhl-Leibniz-ZentrumfürInformatik, 2009.
3. Mohamed, M. "Wireless Semantic RESTFUL Services Using Mobile Agents." Infomesr Org 4: 490-494, 2012.
4. Inoue, Takeru, et al. "Key roles of session state: Not against REST architectural style." Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual. IEEE, 2010.
5. Mihindukulasooriya, Nandana, Miguel Esteban-Gutiérrez, and Raúl García-Castro. "Seven challenges for RESTful transaction models." Proceedings of the 23rd International Conference on World Wide Web. ACM, 2014.
6. Gorski, Peter Leo, et al. "Service security revisited." Services Computing (SCC), 2014 IEEE International Conference on. IEEE, 2014.
7. Serme, Gabriel, et al. "Enabling message security for RESTful services." Web Services (ICWS), 2012 IEEE 19th International Conference on. IEEE, 2012.
8. Viega, John, J. T. Bloch, and Pravir Chandra. "Applying aspect-oriented programming to security." Cutter IT Journal 14.2 (2001): 31-39.
9. Kuhlemann, Martin, and Christian Kästner. "Reducing the complexity of AspectJ mechanisms for recurring extensions." Proc. GPCE Workshop on Aspect-Oriented Product Line Engineering, AOPLE. 2007.
10. Santos, Adriano, et al. "Avoiding code pitfalls in Aspect-Oriented Programming." Science of Computer Programming 119 (2016): 31-50.
11. Harbulot, Bruno, and John R. Gurd. "A join point for loops in AspectJ." Proceedings of the 5th international conference on Aspect-oriented software development. ACM, 2006.
12. Guo, Li-Qing, Kuo-Hsun Hsu, and Chang-Yen Tsai. "A study of the definition and identification of bad smells in aspect oriented programming." e-Business Engineering (ICEBE), 2015 IEEE 12th International Conference on. IEEE, 2015.
13. Khatri, Sunil Kumar, Pankaj Narayan, and Prashant Johri. "Weaving Techniques and its impact on execution of classes in Aspect Oriented Programming." Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), 2015 4th International Conference on. IEEE, 2015.
14. Alves, Péricles, Eduardo Figueiredo, and Fabiano Ferrari. "Avoiding code pitfalls in aspect-oriented programming." Brazilian Symposium on Programming Languages. Springer International Publishing, 2014.