Towards Introducing and Implementation of SOA Design Antipatterns

Deepali Tripathi, Ugrasen Suman, Maya Ingle, and S. K. Tanwani

***Abstract*—Service Oriented Computing (SOC) is an emerging distributed computing technology that is set to replace the existing ways of building software. Dissatisfactory performance of SOA projects has stimulated the developers to analyze the SOA worst practices or antipatterns. Our research aimed at identifying these wrong practices in implementation of SOA, i.e. antipatterns. In this paper, four antipatterns SOA==SOAP, using plain WSDL, web service discovery only through UDDI, and service for an application have been identified and presented in SOA antipattern template. These antipatterns are related to the use of SOAP, WSDL, UDDI and basic service definition, which initially seemed to be correct but later resulted into reduced performance benefits.**

***Index Terms*—Antipatterns, service, service oriented computing, SOA.**

I. Introduction

Service Oriented Computing (SOC) is the latest design paradigm used to implement distributed systems. It comprises of a set of components as services that can be invoked and whose interface descriptions are published and discovered [1]-[3]. The popularity of SOA has motivated designers to document its applications and implementations. Many best practices in the form of design patterns have been defined for SOA. They capture expert knowledge about best practices in software design, in a form that allows that knowledge to be reused and applied in the design of many different types of software. Some of the solutions have stood the test of time while others have not. These blemishing design patterns lead to the concept of Antipatterns.

Antipatterns are specific repeated practices that appear initially to be beneficial but ultimately result in undesirable consequences Documentation of antipatterns helps the programmer to be aware of the common wrong practices, and hence improves the project statistics. There are various problems in adaptation of SOA, which result into the dissatisfactory performance of SOA projects. These problems are to be seriously catered; hence practitioners have started addressing different bad or worst practices of SOA implementation in form of antipatterns.

There are several available SOA best practices and design patterns, which are currently used in the implementation of SOA based projects [3]-[5]. Antipatterns have been addressed by practitioners after year's long experience in the field. A survey on different antipatterns was performed exploring various worst practices and the causes of the failure of SOA projects [6]-[11].

Antipatterns for SOA have already been documented [12]-[16], but our major concern was on the SOA design antipatterns. Moreover, they have been described at different levels of abstraction, which makes them appear independent and isolated. After studying various SOA implementations, case studies [17]-[20], News agency Service project of Signett IT enabled services, Travel Portal project various antipatterns in SOA Design have been identified.

It has been observed that there are some flaws in implementation of SOA. Four antipatterns have been identified in this research work. They concentrate mainly on SOA design. These are SOA==SOAP, Using Plain WSDL, Web service discovery only through UDDI and Service for application. The first antipatterns SOA==SOAP focuses on ignorance of other parallel approaches to SOA. Second Antipattern focuses on improper representation of service using WSDL. Third antipattern recommends the use of REST services which do not require any service registry to be discovered and prefers using customized registries. The fourth antipattern highlights the wrongly implemented concept of service forgetting the basic service design principles.

The rest of the paper is organized as follows. Section II of the paper briefly explains the antipattern template that will be used to describe the proposed antipatterns. Section III explains each of these proposed antipatterns along with their implementation and re-factored solutions. The fourth section provides future work in this direction of research and the conclusion of the paper followed by the references used.

II. SOA Antipattern Template

Antipatterns describe a commonly occurring solution to a problem that generates negative results i.e. seemingly well but in fact, wrong solutions [21]. There are various problems in the adaptation of SOA, which result into the failure of SOA projects. Antipatterns proposed by different organizations have been fragmented and have been focusing on the complete SOA life cycle i.e. from the origin of concept to realization [22-24]. The SOA antipatterns discussed in the next section utilize the following SOA antipattern template to document the common dysfunctional practices in the adaptation of SOA. It specifies the name, root cause, primal forces, description and the name of the antipattern to which the current antipattern is similar to. Like design patterns, antipatterns should also follow a general profile format for

Manuscript received December 30, 2012; revised July 5, 2013.

Deepali Tripathi is with the Modi Institute of Mgmt and Technology, Kota, India (e-mail: deepalidt@gmail.com).

Ugrasen Suman, Maya Ingle, and S. K. Tanwani are with School of Computer Science & IT, Devi Ahilya University, Indore, India.

their representation [9]. Following is the antipattern template used to describe SOA design antipatterns.

Antipattern Name: The Antipattern name is a unique noun phrase. The name is used for future reference to the principles contained in the antipattern. They form the basis for an organization's terminology when members discuss and document software and architectures.

Also Known As: This identifies additional popular or descriptive names and phrases for this Antipattern.

Root Causes: These are the general causes for the antipattern. They can be one or more of the following values:

- Haste: Hasty decisions lead to compromises in software quality. As successive project deadlines are missed, anything that appears to work is considered acceptable, regardless of quality.
- Apathy: It refers to not caring about solving known problems. It is a basic unwillingness to attempt a solution.
- Narrow-mindedness: It is the refusal to practice solutions that are otherwise widely known to be effective.
- Sloth: Automatically generated interface stubs and skeletons make the task of constructing a distributed system relatively easy.
- Avarice: Architectural avarice means the modeling of excessive details, which results in excessive complexity due to insufficient abstraction.
- Ignorance: It is the result of failing to seek understanding. The problem of ignorance (implementation dependency) often occurs in the migration of applications to distributed architectures.
- Pride or responsibility: Often, developers unnecessarily invent new designs when knowledge from preexisting systems, products, and standards are readily applied through architecture mining.

Primal Forces: Forces are concerns or issues that exist within a decision-making context. The choices include any of Management of functionality, Management of performance, Management of complexity, Management of change, Management of IT resources and Management of technology transfer.

Re-factored Solutions. This section explains a re-factored solution that is structured in terms of solution steps.

III. Proposed SOA Antipatterns

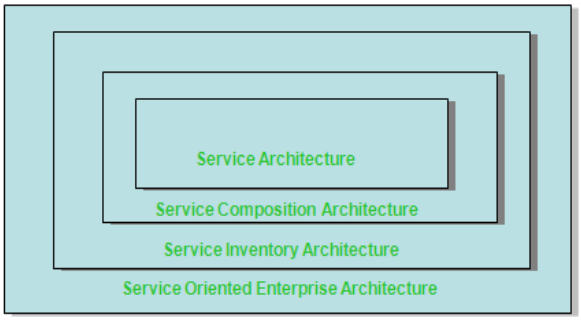

Fig. 1. Layered architectures.

Design patterns are proven solutions to a task presented in a standard format and antipattern are the wrong ways of doing a task which initially seemed to be correct [24]-[27]. SOA comprises of different architectures as shown in Fig. 1. Antipatterns may exist in any of the layers shown, but our research concentrated mainly on the antipatterns related to SOA design.

In this paper, SOA==SOA, using plain WSDL, web service discovery only through UDDI and service for an application are identified as antipatterns and these are discussed in the following subsections.

A. SOA==SOAP

Practitioners implementing SOA often consider that, the three standards required for implementing web services are the Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). SOAP is an XML-based protocol to support communication between a Web service, its clients, and UDDI registry. WSDL is an XML-based standardized interface definition language used to describe what a web service can do, where it resides, and how it can be invoked. UDDI standard is used to publish, discover, and manage web services in an UDDI registry. A REST (Representational State Transfer) web service is basically a simplified model where everything is wrapped around the HTTP send/receive protocol.

Using services based on SOAP envelop always, may be an overhead, whereas that same work could be done using lightweight approach like REST using traditional methods. The main responsibility of accessing the service in the SOAP-WSDL process lies on the consumer.

Although core services holding logic should be bind in an SOAP envelop but simple data handling services , CRUD operations should be implemented using traditional Http methods viz. GET, PUT, POST, DELETE i.e. through RESTful way. REST emphasizes resources with a uniform interface for addressing them, while SOAP based RPC emphasizes processes with a uniform interface for invoking them . With a RPC-based architecture, there is no limit to the number of processes. For services representing basic CRUD operations, REST way of implementing services is simpler and lightweight.

1) Re-factored solution

In the development of Web service based SOA applications, the designing of services should not be adamant to a traditional style but other approaches should equally be used when and where required. SOAP based and REST based have been compared in the following paragraphs and depending on the requirement, appropriate method for implementing services could be selected. Both these approaches are not the counterparts and can be used together in the same application.

a) Approach

Let us first discuss these two different approaches of implementing web services. REST is an architectural style that prescribes the use of standards such as Http, URL, Resource representations (XML, HTML, Gif etc.), MIME Types etc. [11]. The RESTful web service makes available URL to a resource and it may allow the client to specify the format of the returned resource i.e. HTML or an XML

document. The service itself may be described using WSDL or WRDL (Web Resource Description Language) and can be accessed either as a resource or using JSON (Java Script Object Notation). RESTful services are stateless; each request from client to server must contain all necessary information. All resources are accessed with generic interface (Http GET, POST, PUT, DELETE). These resources are named using URI (Uniform Resource Identifier). The client may progress from one state to another using interconnected URL representation.

In SOAP method, provider creates and implements a web service interface on an existing application. He has to create a XSD (XML Schema Document) and WSDL contract in order to distribute the web service details to potential consumers. Consumer obtains WSDL contract for consumption through UDDI registry. It is the responsibility of the SOAP server to a parse the SOAP message and determines which method to invoke. The returned data would not contain any URL, since a URL that points to a SOAP service is just to the SOAP server. In REST all decisions are made based upon the URL and the Http method selected while in SOAP, server receives all messages, peeks into the SOAP envelop and then distributes each message to the appropriate application for processing.

b) Proxy servers

Proxy servers play a major role as web intermediaries for a web application. In the REST approach, the URL identifies the resource that is desired. The Http method identifies the desired operation. The Proxy server decides based upon the identified resource and the Http method whether or not to allow the operation. Using XLink (the XML hyperlink technology) in addition to providing a URL to the target resource, data about the resource could also be provided using Xlink:role. The application can dynamically make decision about what resource is to be accessed next.

In SOAP based approach, proxy server cannot directly allow or disallow the message since it is unaware of the desired contents or resources. Either the proxy server should understand the semantics of each SOAP application that a client will access, but for that the proxy server will need to be updated for each new SOAP application.

c) Caching

It refers to the ability to maintain a copy of the desired resources in order to improve the performance. In the REST approach, the response of a resource contains an indication in the Http header of whether the results are cacheable or not. If it is, cache servers make a local copy, which can be returned for the same request if repeated. A SOAP message is always with a POST method, which makes the cache server unaware of the actual intention of the request type (GET or POST). Moreover the SOAP URI is always to the SOAP server which prohibits the cache server again from knowing the actual resource requested. Hence no caching is possible with SOAP.

d) Generic Interface

Generic interfaces imply generalized functionality and hence support scalability whereas application specific or custom interface interfaces may need some additional functionality to be called in a generic context. In REST, every resource has a generic interface namely Http GET, PUT POST, and DELETE which enable caching and proxy servers to do their work. Whereas in SOAP, There is no defined standard set of methods. Any type of methods could be defined which makes customization on application basis and reduces scalability.

e) Interoperability

Interoperability means sharing the data amongst multiple applications. The more interoperable software programs are, the easier it is for them to exchange information. In REST, Interoperability is based on standardization. REST relies on standards of addressing and naming resources (URI), resource interfaces (GET, POST, PUT etc.), representations (HTML, XML etc.), and media types (MIME types).

REST and SOAP do not replace each other, each of them have their uses but when making high performance and client rich websites REST can provide a significant improvement. Traditional way of implementing SOA only through SOAP also leads to other antipatterns. REST style needs no registry and makes resources directly available hence it also helps in overcoming the following two antipatterns viz. Discovery of web service through UDDI and Using plain WSDL to define service interface.

2) Standard representation

Following the Standard Antipattern Template [14] and SOA Antipattern Template, the above proposed antipatterns can be described as follows:

Antipattern Name: SOA==SOAP

Also known as/ similar to: Not Applicable

Root Cause: The common and fundamental reasons for the problem can be coined as haste, apathy, ignorance.

Primal Forces: These are certain architecture and development related concerns or issues present in most decision making context. They greatly affect the design and development process and in this case it can be management of functionality and management of technology transfer. Misuse of these above mentioned forces leads to the development of this antipattern.

Description: SOAP-WSDL is considered to be the only way of implementing SOA by companies implementing SOA for the first time.

Solution: Although SOAP-WSDL is the established way of SOA implementation through web services but other alternative ways like REST should be equally considered. For CRUD applications RESTful services should be preferred and for application specific services holding core logic SOAP based services should be preferred.

3) Implementation

Following are few screenshots of their implementation i.e. SOAP-WSDL based web service in .Net through Visual Studio 2008 and REST Based web service in java through Netbeans7.0.1 Fig. 2 represents a structure of SOAP based service. It shows various methods which are application specific and need not have a generic structure.

Fig. 3 shows the structure of REST based service. It reflects certain methods like getJSON() to retrieve java script object notation form of data, getXML() to retrieve its XML format.

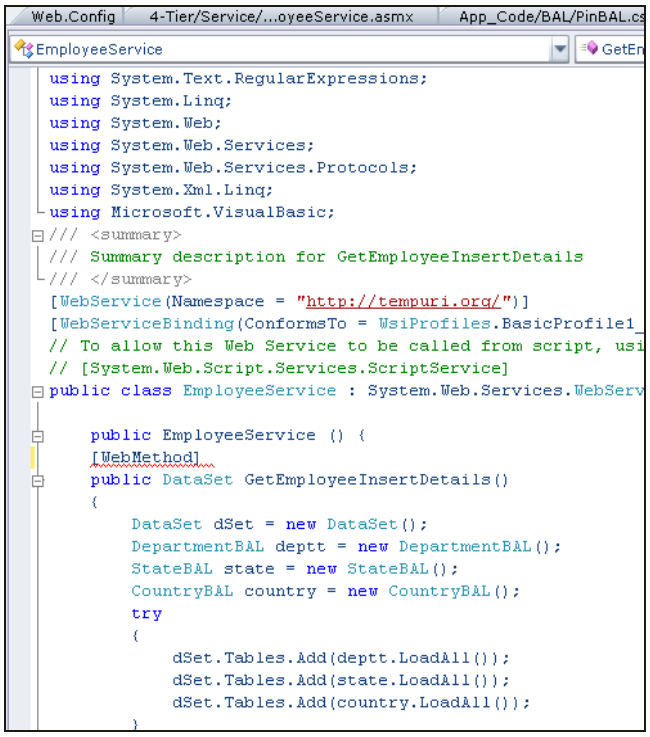

Fig. 2. Structure of SOAP based service.

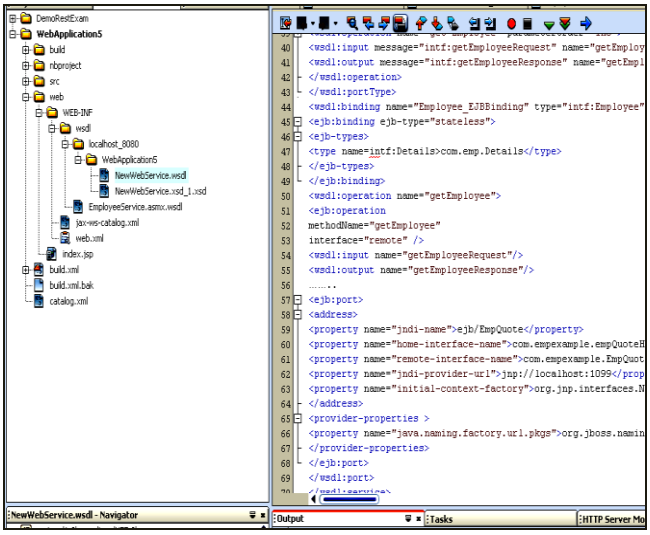

Fig. 3. Structure of REST based service.

Through the interface of the REST based web service the resources are available in the form of URI (Uniform Resource Identifier) in the returned page. User can access these web services by simple clicking on the URL shown, the get XML () or get JSON() methods are called accordingly.

B. Using Plain WSDL

WSDL (Web Service Description Language) is used to define service interfaces. It describes two different aspects of a service: its signature (name and parameters) and its binding and deployment details (protocol and location). WSDL does not contain full interface of a service, it does not have any semantic information [28]. A WSDL file does not specify how to access next desired service, how long a service usually runs, who is allowed to call it, how much a service call cost and many other non functional attributes. All these aspects must usually be known in order to manage a service in a large SOA landscape. With future WSDL versions this might change.

1) Re-factored solution

Service Description should be provided in a separate format and WSDL should be generated from it when required. WSDL files can be extended internally with additional XML elements and attributes or externally with supplementary files [20]. WSDL allows elements representing a specific technology under various elements defined by WSDL. These elements are known as extensibility elements. Extensibility elements allow vendors to expose their Web Services as EJB's, Remote Java Objects and .NET objects without having to write SOAP bindings for them. Currently, the WSDL specification introduces specific binding extensions for the SOAP, HTTP GET/POST, MIME protocols and message formats.

Using the extensibility mechanism a service developer can describe commonly used services such as EJB, .NET and Java Objects. The consumer of the service can use the WSDL and generate the necessary client side stubs to invoke the endpoints in the native protocol. This approach has a several advantages. A service can have multiple bindings associated with it and the consumer of the service will have the choice of selecting one binding or the other.

a) Implementation

The Fig. 4 below shows the standard WSDL file for a simple web service in java.

Fig. 4. Standard WSDL representing a service.

In the Fig. 5 code segment the information for locating the EJB is stored in <ejb:port> section of the WSDL definition and the information for invoking the EJB is stored in the <wsdl:binding> section.

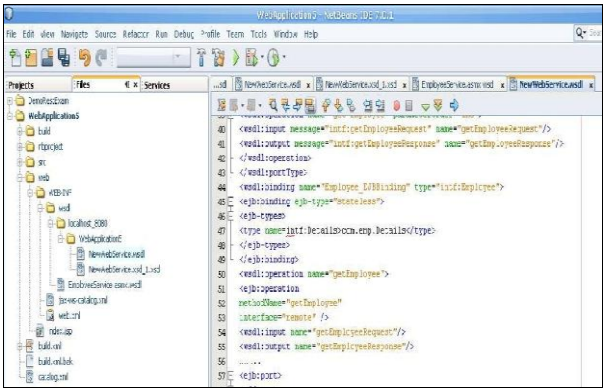

Fig. 5. WSDL extensions using WSDL4J.

2) Standard representation

According to SOA Antipattern Template, the above proposed antipatterns can be described as follows: *Antipattern Name:* Using Plain WSDL to define all service interfaces.

Also known as/ similar to: Not Applicable.

Root Cause: It can be the result of haste, sloth and ignorance.

Primal Forces: Management of change, management of complexity and management of technology transfer. *Description:* Simple WSDL describes only signature (name and parameters) and its binding and deployment details (protocol and location). This does not describe various non functional attributes like how to access next desired service, cost of service etc.

Solution: WSDL files can be extended internally with additional XML elements and attributes or externally with supplementary files. Certain extensibility mechanisms have been defined for specific purposes like, those supported by WSDL4J for ejb's. Techniques for defining WSDL extensions have been proposed [12] and are one of the major research areas in WSDL.

C. Web Service Discovery only through UDDI

In a real SOA enterprise infrastructure with hundreds of services, it is safe to assume that service endpoints are going to constantly be subjected to changes in areas such as location (URL), policy (security, etc) or contract (WSDL, operations).

In order to address these challenges, the big SOA vendors (Microsoft, Oracle, IBM etc.) created a standard that with the purpose of modeling service metadata information that could be used to enable service discovery capabilities. The standard was known as Universal Data Discovery and Integration (UDDI) and, unfortunately, it became the cornerstone of SOA governance products. UDDI has proven to be an incredibly ineffective mechanism to enable service publishing and discovery. The SOA models created with UDDI are incredibly complex to implement and use. They end up becoming another bottleneck of SOA.

1) Re-factored solution

While building SOA application, the complexities of UDDI should be avoided and instead use a simpler mechanism to facilitate the discovery and query of services. This can be achieved by implementing a 100% RESTful API that allows querying the entire service registry using plain HTTP GETs methods. There is no requirement of centralized registry. More advantages of REST are discussed in previous section. User defined or application specific registries can also be defined like Oracle's OSR (Oracle service registry), But these application specific registries are very complex and far from the reach of a simple programmer.

2) Standard representation

According to SOA Antipattern Template, the above proposed antipatterns can be described as follows: *Antipattern Name*: Discovery of web service through UDDI.
Also known as/ similar to: Not Applicable.
Root Cause: It can be haste, sloth and ignorance.

Primal Forces: Management of performance, management of IT resources and management of technology transfer.

Description: Since SOA literatures and previous implementation of the technology, effectively present the usage of UDDI as the central registry for SOA services, the new small projects consider it to be an un-detachable component of SOA. UDDI is incredibly complex and difficult to implement. Even IBM and Microsoft have refrain from their UDDI registries. In such case, adhering to UDDI seems to be right but in fact not the perfect way of service discovery.

Solution: Customized registries according to the application should be created. Various other registries using JNDI (Java Naming and Directory Interface), OSR (Oracle Service Registry) can also be used in an SOA application. REST based services should be preferred for data access services. They are directly accessed through URI's hence require no central registry.

D. Service for an Application

In the development phase of the module it has been observed that the first step in implementation of SOA, if taken mistakenly can prove to be a useless investment. Services are supposed to be designed for achieving main goals of SOA viz. reusability, interoperability, increasing organizational agility etc. Many IT developers with object oriented experience implement SOA in the way they started Object oriented software. Services are designed application specific. No enterprise level service classification is involved. Service just become another way of creating an application, hence, provides no business benefits. Large numbers of services are designed, leading to another antipattern: Service Silos.

1) Refactored solution

Proper training and education of basic SOA goals and principles should be given to the involved members before the actual work begins on the project.

The service design should also follow basic SOA design principles [12]:

1) Standardized Service Contract: Services in the same inventory should follow same design contract.
2) Service Loose Coupling: Services should be loosely coupled with customer requirements and their own surrounding environments.
3) Service Abstraction: Service contract should contain only the essential generic information.
4) Service Reusability: Services should have reusable enterprise logic.
5) Service Autonomy: Services should be autonomous i.e. their runtime environment should be under their control.
6) Service Statelessness: State information should not be maintained with service itself.
7) Service Discoverability: Services should be effectively discovered and interpreted through suitable mechanisms.

2) Standard representation

According to SOA Antipattern Template, the above proposed antipatterns can be described as follows: *Antipattern* Name: Service for an Application.

Also known as/ similar to: Not Applicable.

Root Cause: It can be haste, apathy, sloth and ignorance. Primal Forces: Management of functionality, management of change, management of complexity and management of

technology transfer.

Description: Services are built for use within an application forgetting the basic service design principles.

Solution: The services should be classified as intra application and inter application. Inter application services should be designed for interoperability. Application specific services if required should be at the lowest level and callable only by the generic services providing interface to the service consumer. Services at lowest level should further be properly identified as entity services, task services and utility services [15]. Services should essentially follow basic design principles for a successful SOA implementation.

IV. Conclusion and Future Work

It has been observed that amongst the large number of addressed SOA antipatterns, failures are mainly due to limited number of interrelated antipatterns focusing mainly on the SOA design [29]. Four antipatterns SOA==SOAP, Discovery of web service through UDDI, Using Plain WSDL to define all service interfaces, Service for an application were identified and represented. The above conclusions and derivations were based on the case studies and SOA implementation, using both, SOAP based and REST based services. In this paper we mainly emphasized SOA design antipatterns. Some of the domain areas such as request change, data handling have been left unexplored and few more antipatterns can be identified. A framework for building SOA applications could also be developed which would integrate various features necessary for SOA implementations.

References

[1] L. Srinivasan, "An overview of Service Oriented Architecture, Web Services and Grid Computing," HP (Hewlett Packard) White Paper, November 2006.

[2] Y. Zhao, "Service Oriented Infrastructure Framework," *IEEE Congress on Services*, 2008.

[3] M. P. Papazoglou and P. Traverso, "Service-Oriented Computing: State of the Art and Research Challenges," *IEEE Computer Society*, November 2007.

[4] S. Chatterjee, "An Introductory Overview of Web Services," *CSI Journal*, pp. 6-12, March 2007.

[5] D. Jana, "Service Oriented Architecture-A new Paradigm," *CSI Journal*, pp. 12-15, March 2006.

[6] T. Erl, *SOA: Principles, Concepts and Techniques*, 1st ed. Prentice Hall, 2009.

[7] A. Kontogogos and P. Ageriou, "An overview of Software Engineering approaches to Service Oriented Architectures in various fields," in *Proc. 18th IEEE International Workshop on Enabling Technologies*, 2009.

[8] D. Tripathi, "Development Trends and Evolution of SOA," in *Proc. Emerging Trends in Mechanical, Electronics and Computer Engineering*, April 2010, pp. 139-143.

[9] J. Evdemon, *"Principles of Service Design: Service Patterns and Antipatterns,"* Microsoft Corporation, Architecture Strategy, August 2005.

[10] T. Erl, *SOA: Design Patterns*, 1st ed. Prentice Hall, 2009.

[11] G. Farrow, *SOA Antipatterns*, IBM White paper, June 2009.

[12] T. Erl, *SOA: Principles of Service Design*, 1st ed. Prentice Hall, 2009.

[13] *SOA Antipatterns: How not to do service Oriented Architecture*, Oracle White Paper in Enterprise Architecture, January 2010.

[14] J. Kral and M. Zemlicka, "Popular SOA Antipatterns," *Computation World: Future Computing, Service Computation, Cognitive Content*, Patterns, 2008.

[15] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, 2000.

[16] D. Tripathi, U. Suman, and M. Ingle, "A systematic review of Antipatterns in SOA," in *Proc. Computing Business Application and Legal Issues (ICCBALI'11)*, Ghaziabad, March 2011, pp. 2-7.

[17] M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Krogdahl, M. Luo, and T. Newling, "Patterns: Service Oriented Architecture and Web Services," *IBM Redbook*, April 2004.

[18] J. Chung and M. Zhu, "Evaluating a Service-Oriented Travel Portal," in *Proc. Fifth IEEE International Symposium on Service Oriented System Engineering*, 2010.

[19] Web Services. [Online]. Available: http://www.java.sun.com/techArticles/WebServices/soa3/loanprocess.htm

[20] C. Satish, "Barriers of SOC," in *Proc. the Second Workshop on Introducing Service-Oriented Computing WISOC*, 2007.

[21] S. Moosavi and M. Seyyadi, "A method for Service Oriented Design," presented at 6th International Conference on IT, New Generations, 2009.

[22] N. Milanovic, "Service Engineering Design Patterns," presented at 2nd International Symposium on Service Oriented Systems Engineering SOSE, 2006.

[23] Reference Service-Oriented architecture model 1.0. *Commission Specification 1*. (August 2, 2006). [Online]. Available: http://www.oasisopen.org/committees/download.php/19679/soa-rm-cs.pdf

[24] C. Smith and L. G. Williams, "Software Performance Antipatterns," presented at 2nd International Workshop on Software Engineering and Research, 2008.

[25] SOA Pattens. [Online]. Available: http://www.SoaPatterns.org

[26] S. Chatterjee, "A Introductory overview of Web Services," *CSI journal* vol. 29, issue. 9, pp. 6-12, March 2007.

[27] J. Fronckowiak, "SOA Best practices and design patterns," White paper, March 2008

[28] S. Punita and C. Babu, "Performance prediction model for service oriented applications," in *Proc. 10th International Conference on HPC and Communications*, 2008.

[29] C. Dai, "A flexible extension of WSDL to describe nonfunctional attributes," *IEEE*, 2010.

[30] W. J. Brown and R. Malveau, *Antipatterns: Refactoring Software, Architectures and Projects in Crisis*, 2nd ed. John Wiley, 1998.

Deepali Tripathi was born in Kota, Raj, India. After finishing her masters in Mathematics from MDS University Ajmer. She completed her MCA from Rajasthan Vidyapeeth Udaipur. She did her M.Tech (Computer Science) from Devi Ahilya University, Indore.

She has more than twelve years of teaching experience in College education. She is currently an associate professor at MIMT, Kota Rajasthan. She has published 4 International papers, 7 National papers and 2 Books on programming.

Ms Tripathi is a member of IEEE, ISTD and CSI. She is also a member of review committees and editorial board of few journals.

Ugrasen Suman has received his master degree in Computer Applications from Rani Durgawati University Jabalpur and Ph.D. degree in Computer Science from Devi Ahilya University Indore, India. He is presently an associate professor at School of Computer Science & Information Technology, Devi Ahilya University, Indore. He is having more than 11 years teaching and research experience. His areas of research are Software Engineering, Knowledge Management & Mining, Databases & Information Retrieval, Programming Paradigms and Service Oriented Computing. He has guided One Ph.D. scholar, Four PG research scholars and more than 45 PG projects. Currently he is guiding Eight PhD scholars and Three PG projects. He has published more than 50 research papers. He is also working on a UGC-SAP research project of Data Mining and Software Engineering. He is a member of ACM-SIGSE, senior member of IACSIT. He is a reviewer/referee of computer science journals/ conferences in various publications.