International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:3, 2009

Dynamic Coupling Metrics for Service – Oriented Software

Pham Thi Quynh, Huynh Quyet Thang

***Abstract*—**Service-oriented systems have become popular and presented many advantages in develop and maintain process. The coupling is the most important attribute of services when they are integrated into a system. In this paper, we propose a suite of metrics to evaluate service's quality according to its ability of coupling. We use the coupling metrics to measure the maintainability, reliability, testability, and reusability of services. Our proposed metrics are operated in run-time which bring more exact results.

***Keywords*—**Dynamic coupling metric, SOA, web service, SOAP Extension.

I. Introduction

SERVICE-ORIENTED ARCHITECTURE (SOA) is an approach to build distributed systems by integrating components that have independent platform, language, and operating system. SOA delivers application's functionality as services to end-user applications or to build other services [1]..

SOA is an architecture that uses open-standards to describe software components. SOA provides a standard way for describing and interacting between software components. Specific software components become basic blocks and they can be reused to build other applications.

Software components are called services. Services are important elements in SOA. We need a clear understanding of the term *service.* A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services. Service-oriented system only can operate when services in this system have to collaborate. Collaboration between services in system is described as Fig. 1.

A service provider publishes a service description to a service registry. When a service description is available, a service consumer can find any service via the service registry, the service's description contains sufficient information for the service consumer to bind to the service provider to use it.

Web Service is a technology that is well suited to implementing a service-oriented architecture. In essence, Web services are self-describing and modular applications that expose business logic as services that can be published, discovered, and invoked over the Internet [2]

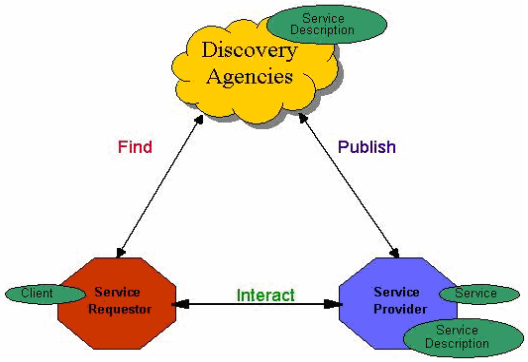


Fig. 1 Collaboration between services in a system

Web services use some standards based on XML. These standards define describing (WSDL), finding (UDDI) and communicating between services (SOAP) in system.

The following describes the sequence of events that occur when an XML Web service is called:

1) The client creates a new instance of an XML Web service proxy class. This object resides on the same computer as the client.
2) The client invokes a method on the proxy class.
3) The infrastructure on the client computer serializes the arguments of the XML Web service method into a SOAP message and sends it over the network to the XML Web service.
4) The infrastructure receives the SOAP message and deserializes the XML. It creates an instance of the class implementing the XML Web service and invokes the XML Web service method, passing in the desterilized XML as arguments.
5) The XML Web service method executes its code, eventually setting the return value and any out parameters.
6) The infrastructure on the Web server serializes the return value and out parameters into a SOAP message and sends it over the network back to the client.
7) The XML Web service infrastructure, on the client computer, receives the SOAP message, deserializes the XML into the return value and any out parameters, and passes them to the instance of the proxy class.
8) The client receives the return value and any out parameters.

Manuscript received June 05, 2009. This work was supported in part by the Vietnam's Ministry of Science and Technology under Grant KHCB2.034.06.

Pham Thi Quynh is a Lecturer at Software Engineering Department, Faculty of Information Technology, Hanoi National University of Education (E-mail: ptquynh@gmail.com).

Huynh Quyet Thang is an Associate Professor at Software Engineering Department, Faculty of Information Technology, Hanoi University of Technology, Hanoi, Vietnam (E-mail: thanghq@it-hut.edu.vn).

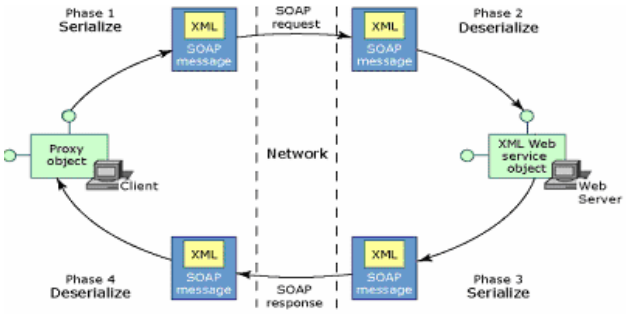


Fig. 2 Web service lifetime

Services in a system need to couple to execute a task. However, this coupling is not always available except to it need to maintain. This loose coupling make SOA is easy when maintenance and operation because it operates in individual module in effect to other modules in the software [1].

Coupling or dependency is the degree to which each program module relies on each one of the other modules. Comparing with traditional system, SOA is looser coupling [3]. This avoids the changes by limiting the impact on other elements which have relationship with changed element.

Loose coupling describes an approach where integration interfaces are developed with minimum assumptions between the sending/receiving parties, thus reducing the risk that change in one application or module will effect to other applications or modules. Loose coupling simplifies testing, maintenance and troubleshooting procedures because problems are easy to isolate and unlikely to spread or propagate. Loose coupling minimizes unwanted interaction among system elements.

Nowadays, some coupling metrics of software have been proposed [10]. However, they only evaluated the coupling of object-oriented software. For service-oriented systems, the coupling metrics, which has been proposed, is static [9,11].

Therefore, in this paper, we will propose a set of dynamic coupling metrics of service-oriented software. In the next sections, we will explain how to implement and user guide for this set of metrics. The last section contains conclusions and further work.

II. THE PROPOSED METRICS

Coupling is the most important attribute of service oriented software. The coupling is presented by relationship between services. This relationship shows dependency between services. If a service has more relationship with other services, it will depend on others much more. When a service need to change which impacts on other services – related to it and reverse. Therefore, if a service has more dependency relationship, the coupling between this service and others will become tighter. We can see from relationship in a service - oriented software generally, the more relationship exist inside software, the tighter coupling attribute is. This kind of relationship or connection between services is a skeleton in service oriented architecture.

We develop a suite of dynamic coupling metrics for service oriented software. The "dynamic" concept can be showed by interaction between services in a system at runtime.

A. Coupling Between Services Metric (CBS)

CBS is built directly from CBO (Coupling Between Objects) metric in a suite of C&K metrics [4]. For service A, CBS metric is calculated based on the number of relationships between A and other services in system.

$$CBS = \sum_{i \neq j=1..n}^{n} A_i B_j \quad (1)$$

In which: n is the number of services in system; $A_iB_j=0$ if A_i does not connect to B_j and $A_iB_j=1$ if A_i connects to B_j

When a developer builds service A, he can design many things such as data, messages, operators, use cases … relate to the functions that service A covers. In fact that we can not use everything which service A provides. For example, a developer thinks that service A can interact with service B, C, D; but in runtime it only communicates with service B. It means that calculating dynamic coupling between services will bring a more exact result than based on design specification.

For a service A, the larger the value of CBS metric, the tighter the relationship with other services is. In other words, service A depends much more on others. If these services change which affects on service A, the maintainability will be low.

B. Instability Metric for Service Metric (IMS)

According to METRIC ADVISOR from C-DAC [5], fan.in metric of function A is calculated by the number of functions that call to function A and fan.out metric is the number of functions that are called by function A.

The fan.in and fan.out metrics are used to evaluate software's maintainability. Fan.out metric shows the number of functions that function A calls to; therefore these functions change which make function A also changes. In short, the cost of maintain for a function which has high fan-out metric's value is very high.

Fan.in metric is the number of functions that call to function A. The fan.in metric's value of function A is high means that there are many functions that use and depend on it.

Joost Visser proposed a formula for calculating instability of a component based on fan-in and fan-out metrics [6].

$$\frac{fan.out}{fan.in + fan.out} \quad (2)$$

From above definitions, we built a metric which shows interaction between a service and others in system through sending and receiving messages. Considering service A, we suppose fan-in metric is calculated by the number of messages which are sent to service A and fan-out metric is the number of messages which are sent by service A. Next, we apply the formula for calculating instability of a service.

$$IMS = \frac{fan.out}{fan.in + fan.out} \times 100\% \quad (3)$$

If the value of this metric is low, the level of dependency of

service is low whereas others depend on it higher. IMS = 0 means that the stability of service is very high. IMS = 1 means that the service is very instable.

C. Degree of Coupling between 2 services metric (DC2S)

DC2S metric is developed from CBS metric. As presenting in 2.1, CBS metric is only applied on all of services in system. The DC2S metric identifies relationship between two services to detect the dependency between these services.

Considering service A and service B, the DC2S metric between A and B is calculated by the percentage of the number of times from A to B in the number of times from A to other services in system.

$$DC2S = \frac{N(A,B)}{\sum_{i=1}^{n} N(A,B_i)} \times 100\% \quad (4)$$

In which, n is the number of services in system; $N(A, B_i)$ is the number of connections from service A to service B_i (which is the number of calls from A to operators of B_i)

DC2S metric identifies the level of coupling between two services in runtime; for example, in specification service A has relation to service B and service C. However, in runtime, A calls to B by 100 times, whereas it only calls to C by 3 times. This shows that service A couples with service B tighter than service C. From this point, when maintaining service A, we should concentrate the level of impact of service B higher than service C.

D. Degree of Coupling within a given set of services metric (DCSS)

For a service oriented system, we can build a graph G(V,E) which can describe services and relation between them in this system [6], in which: (i) Nodes in graph is services; (ii) Relation between services is edges; (iii) Direct of edge is direct from request service to provider service and (iv) Weight of edge is identified in two cases. In first case – static metric, weight of edge is 1. In second case – dynamic metric, weight of edge is the number of times from request service to provider service.

After that, we build distinct function from node u to node v [7]. We call d(u,v) is the length of shortest path from u to v. If between u and v does not exist any paths, in theory d(u,v) is ∞, but we choose d(u,v) is K in which K is the maximum value in the length of shortest path between any two nodes. In static metric, K is the number of nodes in graph. In dynamic metric, K is the sum of edge's weight.

For example, a system contains services A, B, C and D (See Fig. 3). If service A calls service B by one time and service A calls service C by one time and service B calls service C by one time and service C calls service D by one time, we will receive a below matrix as shown in Fig. 4.

Next, we will replace ∞ by the sum of edge's weight, means that K is 4.

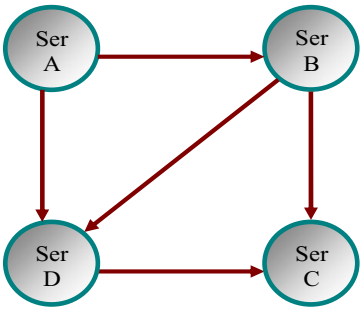


Fig. 3 A system with services and relation between services

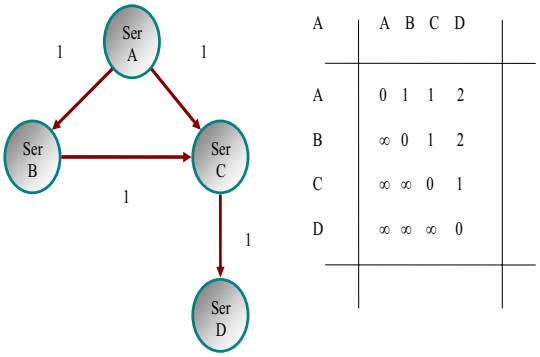


Fig. 4 Graph and distinct matrix

To evaluate the ability of coupling with a service which belongs to the given services, we will build a graph and a matrix for this given services. We can see that the ability of coupling of a service is the level of easy to reach a node in graph [7].

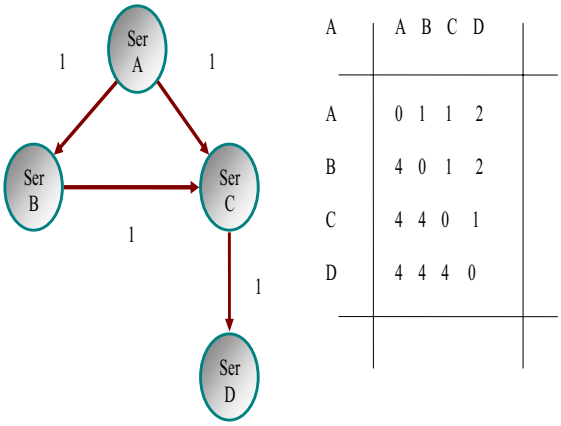


Fig. 5 Graph and distinct matrix with K=4

From this point, we continue to develop a formula for DCSS metric:

$$DCSS = \frac{Max - \sum_{u \in V} \sum_{v \in V} d(u,v)}{Max - Min} \quad (5)$$

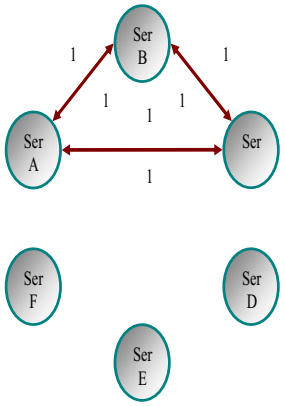
In which: Max only appears when all of nodes in graph do not connect together. Max = K*V*(V-1)

Min only appears when all of nodes in graph connect to others. Min = V*(V-1)

DCSS reflects the coupling between services in a system. If this coupling is loose (means that a service can self provide every requirements), the system will be easy and cheap in maintenance and flexible with change.

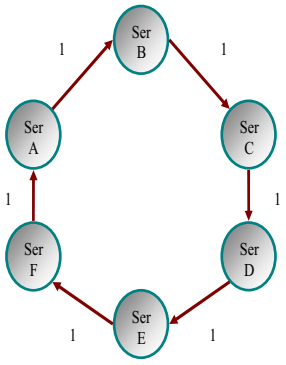
Based on definition of DCSS metric, the value of this metric is low, the coupling in system will be loose and reserve. This metric helps to distinguish the difference between two systems which have the same nodes but differ in the connection between nodes.

In Fig. 6, both (a) and (b) have K = 6 and V = 6 so that Min = 30 and Max = 180. In (a), DCSS = 0.2 but in (b) DCSS = 0,6. This proves that (b) has higher coupling.



	A	B	C	D	E	F
A	0	1	2	3	4	5
B	5	0	1	2	3	4
C	4	5	1	2	3	4
D	3	4	5	0	1	2
E	2	3	4	5	0	1
F	1	2	3	4	5	0

Fig. 6 (a) K=6; V=6; DCSS = 0.2



	A	B	C	D	E	F
A	0	1	2	3	4	5
B	5	0	1	2	3	4
C	4	5	1	2	3	4
D	3	4	5	0	1	2
E	2	3	4	5	0	1
F	1	2	3	4	5	0

Fig. 6 (b) K=6; V=6; DCSS = 0.6

Fig. 6. An example about 2 systems which have same nodes and different edges

TABLE I
A SUITE OF DYNAMIC COUPLING METRICS

METRIC	FORMULA	MEANING
CBS	$CBS = \sum_{i \neq j=1..n}^{n} A_i B_j$	Measuring the coupling between services.
IMS	$IMS = \frac{fan.out}{fan.in + fan.out} \times 100\%$	Measuring the instability of service
DC2S	$DC2S = \frac{N(A,B)}{\sum_{i=1}^{n} N(A,B_i)} \times 100\%$	Measuring the level of coupling between tow services in a system.
DCSS	$DCSS = \frac{Max - \sum_{u \in V} \sum_{v \in V} d(u,v)}{Max - Min}$	Measuring the coupling in a suite of given services.

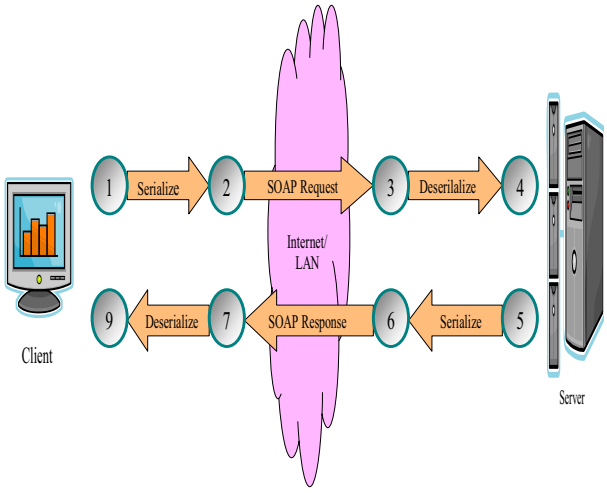


Fig. 7. Communication of Web-service when works with SOAP extension

III. IMPLEMENTATION

To apply proposed metrics for web service, the most importance is to get information about the number of services request to a service that needs to measure or the number of services that this service request to them, and the number of protocols are requested, and all tasks must be executed when system is operating. To get this information, we used SOAP Extension. SOAP Extension is a mechanism that allows us to

get information about SOAP messages which to communicate between a web service server and a web service customer [8].

Fig. 7 describes positions that we can insert a SOAP Extension into the web service architecture (circles are numbered). As you can see, SOAP extensions are quite flexible: you can run code in a SOAP extension before or after a serialization or deserialization operation.

SoapMessageStage.BeforeSerialize happens just before the SOAP message is serialized. When a SOAP message is processed in client mode (the SOAP message is outgoing), the BeforeSerialize stage occurs immediately after a client invokes a Web service proxy method, but before the message is sent out over the wire. When the message is being processed in server mode (the SOAP message is incoming), BeforeSerialize occurs immediately after the WebMethod returns and before the return values are serialized and sent back to the client.

SoapMessageStage.BeforeDeserialize occurs before a message is deserialized and turned into a CLR object. When processing in client mode, BeforeDeserialize occurs after receiving the response from a WebMethod invocation, and just before the response is deserialized into a CLR object. When processing in server mode, BeforeDeserialize occurs after a SOAP message is received by the Web server, but before the SOAP message is deserialized into objects and passed as arguments to the WebMethod.

SoapMessageStage.AfterDeserialize occurs immediately after a SOAP message is deserialized into objects. When processing in client mode, the AfterDeserialize stage occurs after the response from the WebMethod invocation has been deserialized into an object, but prior to the client receiving the deserialized results. When processing in server mode, AfterDeserialize occurs after the request is deserialized into objects, but before the method on the object representing the Web service method is called.

SoapMessageStage.AfterSerialize occurs just after a SOAP message is serialized, but before the SOAP message is sent over the wire. When processing in client mode, AfterSerialize occurs after a client invokes a WebMethod on a client proxy and the parameters are serialized into XML, and before the SOAP message is sent over the wire. When processing in server mode, the AfterSerialize stage occurs after a WebMethod returns and values are serialized into XML, and before the SOAP message is sent over the network.

Therefore, to get information about SOAP when a service is server, SOAP Extension listens at positions 4 and 5. In situation which service is client, to get information, SOAP Extension listens at positions 1 and 8.

SOAP Extension code does not change the code of services, because it is packed as a library file and absolute dependent with the code of services. Other words, results response from this code are text files and are verified by service management before they are sent to measure. So implementing metrics will include two main parts:

In first part, we implemented SOAP Extension to get information about SOAP messages and services. SOAP Extension is built as a library file (.dll). Results from the first part are .dat files that contain information need to measurement. These files will be sent for us to measure by DynamicCouplingMetrics tool, the result of measurement will be sent back to customer that requests measuring or stored in database.

In the second part, after received .dat files, we will execute measuring on DynamicCouplingMetrics tool, this tool is implemented to measure dynamically complete, results are viewed as multiform: number values, chart and table format, on the other hand results are stored in Database to follow and statistical.

IV. Experiments and Evaluation

We experienced the suite of dynamic metrics in a real system which is described in Fig. 8. The system contains 4 services: BankNode (BN), ClientNode (CN), ManufactureNode (MN) and SuppliverNode (SN).

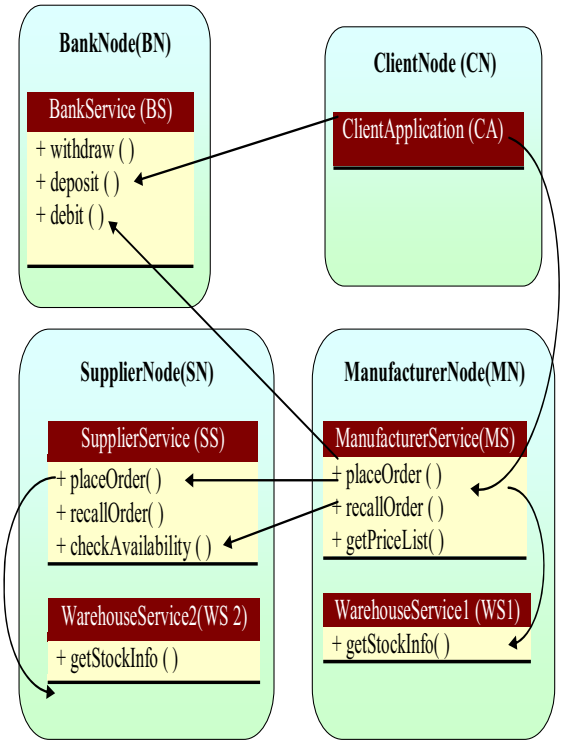


Fig. 8. An example about Web-service system

The result was compared to ARSD metric proposed by Taixi Xu, Kai Qian and Xihe [9]; some metrics for service from Dmytro Rud, Andreas, Schmietendorf, Reiner R.Dumke [10]. Table II shows meaning and measuring methods of these authors.

The suite of our metrics has some advantages. Firstly, it needn't use system's specification whereas almost metrics which were proposed before need to be provided design specification of BPEL file.

TABLE II
METRICS FOR COMPARING

METRIC	FORMULA	MEANING
AIS[s]	The number of services which depend on measured service s.	Measuring the level of confidence
ADS[s]	The number of services which are called by service a but do not stay the same position with s.	Measuring the level of confidence
ACS[s]	$ACS[s] = AIS[s] \times ADS[s]$	Measuring the level of confidence
ARSD	$ARSD = \frac{1}{n}\sum_{i=1}^{n} R_i$	The average of dependence

TABLE III
COMPARING 3 METRICS WITHOUT WEB REFERENCE

	CBS(1)	IMS(1)	AIS
BS	0	0	2
MS	2	2/3	1
SS	0	0	1
WS1	0	0	0
WS2	0	0	0
CA	2	1	0

TABLE IV
COMPARING 3 METRICS WITH WEB REFERENCE

	CBS(1)	IMS(1)	AIS
BS	0	0	0
MS	3	2	2
SS	1	0	0
WS1	0	0	0
WS2	0	0	0
CA	2	2	0

Secondly, these metrics measure services that stay in the same position. This leads to limitation in measuring services. Our metrics solve this problem by users can choose any services in any positions.

Thirdly, result from metrics of these authors is fix because of based on static file (design specification of BPEL file). Our proposed metrics measure services in runtime so that result can change according to real cases.

The result from above example in case not considering and considering services stay the same nodes are presented in the table III and table IV.

The communication between services in a node is insignificant which does not affect other service in other nodes. However, if this communication is significant, metrics of other authors can not give a view about system generally.

Without considering the relation between services in a node, our proposed metrics give result like other metrics. Table III shows BS service has CBS = 0 which means that it does not depend on any services, AIS = 2 presents the number of services depend on it is maximum. Both CBS and AIS prove BS service's stability is the best. In contract, CA service has CBS = 2 means that none services call it and AIS = 0 which shows none services depend on it. To sum up, CA service's stability is the lowest. The CBS metric also shows that the stability of MS service is as bad as CA service, but the AIS metric can not conclude like that.

In table IV, both CBS and ADS metrics conclude that the stability of MS and CA services is low. In addition, the CBS metric also asserts MS service is the most instable like the ACS metric gives.

Now we will compare proposed DCSS metric to ARSD metric. From Fig. 6, we can see the coupling of (a) is looser than (b), the DCSS metric proves it but the ARSD metric concludes the coupling of (a) and (b) is same.

Finally, because the proposed metrics are based on connections in runtime, so they should be evaluated when some services can not be connected together or services are out of work.

V. FUTURE WORKS

We will develop a large storage which contains available information of services. This data is result from many times of measuring a service in different times and in different systems. This data will be compared to each others and conclusion from users of service. After that, we can evaluate service exactly. SOAP extension technology provides information about time sending/receiving SOAP message, the size of message and etc. Therefore, in future, we will continue to build a suite of performance metrics.

REFERENCES

[1] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, August 04, 2005.
[2] http://www.w3.org/TR/
[3] Jeffrey Hasan, Mauricio Duran. *Expert Service-Oriented Architecture in C# 2005*. Apress Publisher, Second Edition.
[4] Stephen H. Kan.. *Metrics and Models in Software Quality Engineering*. Addison Wesley, 2002, .
[5] http://www.cdac.in/html/ssdgblr/metric.asp
[6] Joost Visser, Departamento de Inform´atica Universidade do Minho Braga, Portugal, "Structure metrics for XML schema". www.di.uminho.pt/~joostvisser/publications/StructureMetricsForXMLSchema.pdf
[7] Rodrigo A. B., Ehud Rivlin, and Ben Sheneiderman. *Structural Analysis of Hypertexts: Identifying Hierarchies and Useful Metrics*. ACM Trans. Inf. Syst., Vol. 10, No. 2. (April 1992), pp. 142-180.
[8] A Kalani, E Tittel, P Kalani, Que Certification Indianapolis, Ind, "Developing XML Web Services and Server Components with Visual C#. NET and the. NET Framework**",** Que Publishing, 2003 .
[9] TaiXu, Kai Qian, Xi He. *Service Oriented Dynamic Decoupling Metrics*. Computer and Information Science, 2006. ICIS-COMSAR 2006. 5th IEEE/ACIS International Conference on , pp 44-47, 2006.
[10] Dmytro Rud, Andreas Schmietendorf and Reine.. Resource Metrics for Service-Oriented Infrastructures". www.cs.uni-magdeburg.de/~rud/papers/Rud-13.pdf
[11] Huynh Quyet Thang, Pham Thi Quynh, Tran Quoc Viet. The Reusability and Coupling Metrics for Service-Oriented Software. Proceddings of Japan-Vietnam Workshop on Software Engineering 2007, Hanoi September 26-27, 2007, pp. 53-63