An Optimal Web Services Integration Using Greedy Strategy

Wen Ouyang, Min-Lang Chen
Department of Computer Science and Information Engineering
Chung Hua University, Hsin Chu, Taiwan 300, R.O.C
{ouyang , m09502037}@chu.edu.tw

Abstract

Due to the fast advancement of network technologies, the study of service-oriented architectures (SOA) has attracted much attention recently, taking advantage of the benefits of distributed computing and integration. Web service is a very popular and widely accepted implementation of SOA. The use of Web services holds the advantages of a loosely-coupled system where all components can be developed at independent platforms and be connected with the Web service protocols via the network. However, the tendency for methods of applying Web services over the network is getting more and more complex. Many applications rely on not just one Web service, but a whole school of them. Thus, how to compose and integrate different Web services efficiently to provide complicated network services has become an essential topic in system development and design. This paper proposes a new problem which investigates the possibility of minimizing the number of hops of Web services while trying to finish a set of tasks. It also provides a polynomial-time, optimal Web service integration method using greedy strategy to integrate the Web services in order to complete those tasks. This method can achieve the goal of using the minimum number of hops of Web services over the network.

Keywords—Web services, service-oriented architectures, service integration, service composition.

1. Introduction

Recently, due to the fast and prosperous advancement on network technologies, software execution is no longer restricted on only one machine, and the software structure has progressed from simple client/server, 3-tier, to n-tier model-view-controller (MVC) architectures. Therefore, services-oriented architecture (SOA) [1][2] was proposed to conveniently manage and maintain such huge and complicated systems.

SOA is a standard-based open architecture. Its uses the most advanced technologies to achieve the goal of system integration and distributed computing. Some distributed application systems apply remote procedure calls (RPC) to access services on other machines. Some use the Distributed Component Object Model (DCOM) from Microsoft, Remote Method Invocation (RMI) from SUN Corporation, or Common Object Request Broker Architecture（CORBA）from OMG. All these technologies share common characteristics: They are each tied up with specific operating systems or software, and they each use different communication protocols or data formats. Thus, they cannot work with each other seamlessly. Besides, they require a much higher level of technology support.

Web service is one way to implement SOA. The reason Web services receive so much attention is because of their loosely-coupled distributed architectures and they are platform-independent network services. Application software can, through a uniform resource locator (URL), access services on any computer; no matter what type of computer it is; which language it's written in; what operating system is used or which application type it is. As long as the public standard protocols are followed, the two sides can communicate with each other and the users can receive the services they need. Business partners can integrate their processes easier through pre-established service components. Another advantage is that there are more well-designed tools to generate and apply Web services and to repackage existing modules to form new Web services, which makes the work of developing Web services much easier. Many major software vendors [3][4][5][6] have also endorsed the use of Web services, which makes them more popular.

Web service uses a set of protocols based on

978-0-7695-3473-2/08 $25.00 © 2008 IEEE
DOI 10.1109/APSCC.2008.174

Extensible Markup Language (XML): Simple Object Access Protocol (SOAP) [8], Web Service Description Language (WSDL)[9], and Universal Description, Discovery and Integration (UDDI)[10]. WSDL is used to describe the related information about Web services. UDDI publishes the locations and corresponding information of Web services for users to look for services which can satisfy their needs and then request the those services. SOAP defines the runtime messages needed for service request and response. Web Services architecture consists of three primary roles: service provider, service consumer, and the mediator called service broker. Their relationship can be depicted as in Fig. 1.

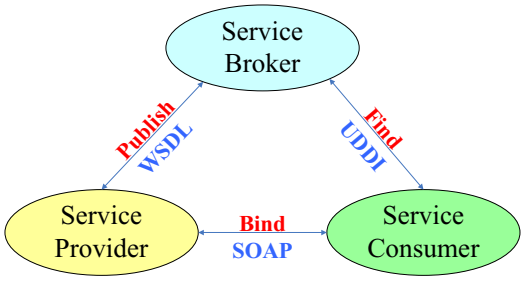

Figure 1 Web services architecture.

Nowadays, more and more researches are focused on service-oriented architecture [2][7], especially on applying Web services. Many of them are discussing how to compose these Web services, or how to apply these Web services. Some studies are investigating the Web services' nonfunctional properties; for example, how to make Web services secure, how to provide highly reliable Web services, and how to evaluate Web services' performance.

Following the trend of complicated application requirements and the fast advancement of Internet technologies, the issue with applying Web services is not just how to find a single Web service which can accomplish one single function. It's essential that we need to satisfy the need of applications with complex requirements. Thus, how to dynamically integrate different Web services to satisfy the changing world and achieve better performance is an important problem to solve. This paper tries to investigate the Web service integration issues, which consider how to choose a set of web services which can satisfy the functional requirements of a pool of tasks and also reduce the Web service change over time. The idea is based on the concern that the communication time used when hopping between Web services contributes most for response time for most of the time. To reduce the hop count between Web services can effectively reduce the change over time.

We propose a novel problem of Web service integration; that is, how to integrate the existing Web services in order to satisfy a pool of tasks and, at the same time, to minimize the Web service hop count. This problem plays important roles when the response time is an important factor in evaluating the performance of the application. We also provide an optimal solution to this problem when the tasks are required to be executed in a specific order.

Section 2 of this paper discusses related work. Section 3 illustrates the proposed polynomial-time, optimal method to solve the Web services integration problem. Detailed data structure illustrations for an example are provided in Section 4. Section 5 concludes this work.

2. Related work

Due to the network technology development, network bandwidth and speed have grown greatly. The Web service technologies are widely researched, applied, and integrated. How to efficiently manage the web services has been studied. Guan, Jin, Wei, and Chen [11] proposed a web services management structure named FASWSM to handle this issue. Web services can be plugged into the application server as a web service adapter to reduce the web services management efforts.

Some researches explore the method of combining different Web services to satisfy complicated function requests. Cheng, Hung, Chiou, and Chang [12] proposed an approach to integrate existing Web services and implement a Web Service Integration Tool for users who are not familiar with programming skills. Users can compose new services by combining the existing service modules. Mohanty, Mulchandani, DeepakChenthati, and Shyamasundar [13] uses finite state machines to modulize different Web services in order to integrate them to new and more complex network services.

Since Web service architecture poses open structures to the outside world, security problems are introduced. Many studies investigate them. Tang, Chen, Levy, Zic and Yan [14] presented a comprehensive performance evaluation to understand the security issue of Web services. Sidharth and Liu [15] presented a framework for enhancing Web services security to prevent DoS or DDoS attacks when designing and implementing Web services. Gutiérrez, Fernández-

Medina and Piattini[16] proposed a Web services Security mechanism as a process that adds a set of specific stages into the traditional phases of WS-based systems development to provide more security. Others like Wu and Weaver[17] proposed, for federated trust management, a framework to facilitate security token exchange across autonomous security domains. J. Li. B. Li, L. Li and Che [18] presented a policy language for adaptive Web services security framework and propose a mixing reasoning framework based on rule and ontology.

Besides the security concern, other Web services' non-functional properties have also drawn attentions. Abramowicz, Kaczmarek and Zyshowski [19] discussed the Web services' reliability from the perspectives of both providers and clients. Zo, Nazareth and Jain [20] provided a basis to measure reliability of an application system that is integrated using Web services. Pat. Chan, Lyu and Malek [21] surveyed and addressed applicability of replication and design diversity techniques for reliable Web services and proposed a hybrid approach to improving the Web services availability.

However, in some applications, the non-functional concerns for composing or integrating web services move away from reliability [20]. In this paper, we find a new viewpoint from which to investigate the Web service integration. In the situation of completing the job with many tasks, those tasks can be supported by some Web services. The issue is when the response time plays an important role; we might want to find a set of Web services which can not only finish all the tasks, but also finish them as quickly as possible. Since Web services are connected over the Internet, the network communication speed contributes most to the response time in most business applications. Thus, the first step to minimize the response time is to minimize the Web service hop count while executing the tasks. The idea is that if there are tasks which can be executed within the same Web service, the communication time between Web services is saved.

This paper also provides a polynomial-time, optimal greedy solution to integrate the Web services in order to achieve the goal of minimizing the hop count of Web services.

3. An optimal greedy algorithm

This work deals with the situations which satisfy the following:

1) All tasks' execution time is negligible compared with network communication time. We also assume that different Web services are located on different computing node over the network.
2) Each Web service is capable of relaying the task requests to the next Web service. This can further speed up the application finish time since each Web service can propagate its results to the next Web service, instead of passing the result immediately to the application if possible. However, even in the situation that the application requires those results right away, to minimize the Web service hop count can still facilitate the shortening of response time.
3) All Web services are located on different machines and require a network to communicate with each other; otherwise we can consider them as the same Web service with different functionalities.

The first problem considered is for the case where there exist no precedence relationships between the unfinished tasks. This problem is equivalent to the problem of using a minimum number of Web services to cover all the tasks. It is also theoretically equivalent to the minimum test cover problem and can be proved to be NP-complete with the minimum set cover problem reduction [22]. Many heuristic algorithms [23][24][25][26][27] [28] have been proposed for the minimum test cover problem, and a recent work in test suite minimization problem using concept analysis [28] can also be applied directly to solve this problem.

The second problem we are focusing on solving is what can be done and how good they are when the tasks have to be finished in a specified linear order, Linear Order Web Service Integration Problem (LOWSIP). That is, the tasks can be sorted in the order of which they need to be finished. Task i has to be done before task j can proceed if and only if $i < j$. In the case of any two consecutive tasks called task i and task $i+1$, if these two tasks are placed on different Web services, then the Web service assigned to execute task i needs to carry over information to the Web service that is to execute task $i+1$ via SOAP protocol. In other words, if two consecutive tasks are assigned to the same Web service, then there is no information passing between web services.

We provide an optimal algorithm, OPT, to find the way to integrate the web services using greedy strategy in the sense of minimizing the hop count between Web services in order to speed up the response time. The OPT algorithm works as follows:

1) Divide each Web service WS_i into the minimum number of mini web services ws_{i1}, ws_{i2}… ws_{ij} where each mini Web service can serve a set of consecutive tasks. Each mini Web service contains trio information: its original Web service, the starting task ID, and the last task ID.
2) Sort the mini Web services in the increasing order of the first tasks they serve. If two mini Web services share the same starting tasks, then remove the one with the smaller last task ID. Keep the leftover mini Web services in an ordered mini Web Service (OMWS) list.
3) Loop steps 4 and 5 through all the tasks starting from task one. Let the current task be task *i*.
4) Task *i* is assigned to the first mini Web service ws_{kl} remaining in the OMWS list.
5) Update the OMWS list by doing the following: (a). Change the start task ID of all mini Web services remaining in the list to the (last task ID of ws_{kl} +1) in case the start task ID is less than the last task ID of ws_{kl}. If a Web service has its last task ID less than its start task ID, remove that Web service from the OMWS list. (b) Remove ws_{kl} from the list.
6) The true Web service assigned to each task can be retrieved from the first part of the mini web service trio information.

An example is used to demonstrate the concepts and relationships between the tasks and Web services of OPT algorithm. In figure 2, the original task and Web service correlations are shown. It shows, for instance, that Web service W_1 can serve tasks T_1, T_3, and T_4.

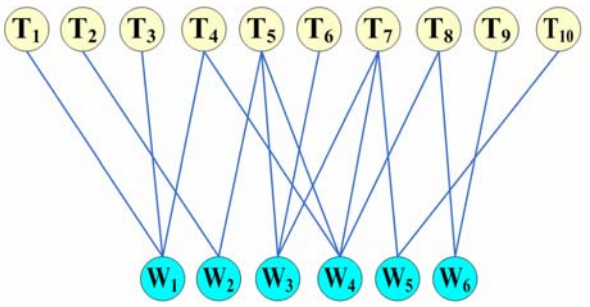

Figure 2 Task & Web services relationship as an example.

Figure 3 shows the result of the Web services' breaking up into mini Web services, followed by figure 4 which exhibits the relationships after the sorting of all mini Web services. Web service W_1 is breaking up into mini Web services ws_{11} and ws_{12}, where ws_{11} services T_1, and ws_{12} services T_3 and T_4. The final result can also be viewed from Figure 4 which reveals the Web service serving each task. Thus, to complete all tasks T_1 to T_{10}, the Web service sequence used for the tasks are WS_1, WS_2, WS_1, WS_1, WS_3, WS_3, WS_3, WS_6, WS_6, WS_5. The number of Web Service hops for this example is five. The result is an optimal solution which means any solution other than this one would have at least five Web service hops.

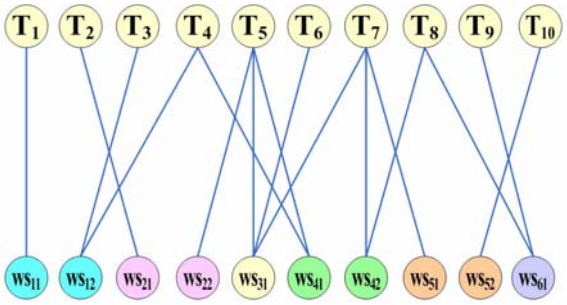

Figure 3 Task & mini-Web services relationships.

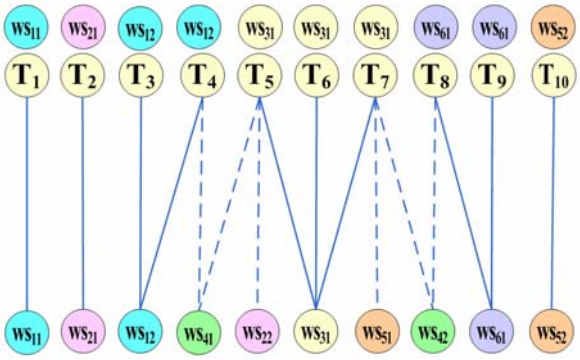

Figure 4 Task and Web service assignments.

The OPT algorithm can be proved to be optimal. Assume that the solution generated from OPT algorithm, S_{opt}, is not optimal for a tasks-Web services scenario. Then, there exists a real optimal solution, say S_r, which leads to a minimum number of Web service hops while S_{opt} does not. That is, this real optimal solution can divide the tasks into fewer chunks than S_{opt} can, where each chunk of tasks is a set of consecutive tasks which are served by the same Web service. Let's compare the first chunks in both S_{opt} and S_r. Since OPT algorithm always picks up a Web service which supports the current task and supports the most other remaining tasks, the size of the first chunk in S_{opt} must be larger or equal to the size of that in S_r. So, we can change the first Web service in S_r to the first Web service in S_{opt} without affecting the

number of Web service hops for S_r. This process continues and we find out that S_r can be completely converted into S_{opt} without increasing the number of Web service changes, so there is no way for S_r to have a less number of Web service changes than S_{opt}. Thus, we prove that OPT produces an optimal solution to have the least number of Web service changes. This algorithm requires polynomial time to complete the task assignment work.

4. A detailed data structure illustration

We use another example to illustrate the details when the data structures used between web services and tasks is a Boolean matrix. In Table 1, there are 20 tasks and 10 Web services, and the matrix illustrates the supporting relationships between them. As in the table, Web service W_1 supports T_2, T_3, T_6, T_8, T_9, T_{10}, T_{15}, T_{16}, T_{17}, T_{18}, and T_{19}

TABLE 1. WEB SERVICES & TASK RELATIONSHIP

Web services supporting relationships
W_1 (0,1,1,0,0,1,0,1,1,1,0,0,0,0,1,1,1,1,1,0)
W_2 (1,1,0,0,0,1,1,0,1,1,1,0,0,1,0,0,1,1,0,0)
W_3 (0,0,1,1,1,0,1,1,0,1,0,1,1,1,0,0,1,1,1,0)
W_4 (1,1,1,0,1,1,1,0,0,0,1,1,0,0,1,1,0,0,1,0)
W_5 (0,1,1,1,0,1,0,1,1,1,0,0,0,0,0,1,1,1,1,0,0)
W_6 (1,1,0,1,1,1,0,1,1,1,1,1,1,0,0,0,0,1,1,1)
W_7 (1,0,0,0,1,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,0)
W_8 (0,0,1,0,1,1,0,1,1,1,0,0,0,0,1,1,1,1,0,0,1,1)
W_9 (0,1,1,1,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,1)
W_{10} (1,1,0,0,1,0,0,1,1,0,0,0,1,1,1,1,0,0,0,1,1)

Each Wi represents a Web service and the boolean values following Wi are the supporting conditions between this Web service and all tasks. 1 means that Wi supports the corresponding task; 0 means Wi doesn't support that task.

The Web services are divided up into mini web services, each of which supports maximum number of consecutive tasks. The next step is to sort the mini Web services according to their starting task, and the result is in Table 2. After that, the mini Web service with the smaller last task ID is removed, if two mini Web services shares the same starting tasks. Also, after each assignment, the OMWS list is updated.

The OPT algorithm then selects the best mini web services and updates the OMWS list until all tasks are satisfied. The final updated OMWS list is shown in Table 3. The Web service assignment result is in Figure 5.

Starting from task T_1 , OPT algorithm picks up the mini Web Services in the OMWS list which support T_1 and also support most other tasks, that is, $ws_{4.1}$. Thus, tasks T_1~T_3 will be handled by $ws_{4.1}$. The next step is to update OMWS list. After that, the algorithm works by starting from task T_4, and the mini Web service left which supports T_4 and is with the most tasks, $ws_{6.2}$, to support will be picked to execute T_4, and so on.

TABLE 2 TASKS & MINI-WEB SERVICES RELATIONSHIP AFTER SORTING

Start task	Mini-web services
T_1	$ws_{2.1}$(1,2), $ws_{4.1}$(1,3), $ws_{6.1}$(1,2), $ws_{7.1}$(1,1), $ws_{10.1}$(1,2)
T_2	$ws_{1.1}$(2,3), $ws_{5.1}$(2,4), $ws_{9.1}$(2,4)
T_3	$ws_{3.1}$(3,5), $ws_{8.1}$(3,3)
T_4	$ws_{6.2}$(4,6)
T_5	$ws_{4.2}$(5,7), $ws_{7.2}$(5,7), $ws_{8.2}$(5,6), $ws_{10.2}$(5,5)
T_6	$ws_{1.2}$(6,6), $ws_{2.2}$(6,7), $ws_{5.2}$(6,6)
T_7	$ws_{3.2}$(7,8)
T_8	$ws_{1.3}$(8,10), $ws_{5.3}$(8,10), $ws_{6.3}$(8,13), $ws_{8.3}$(8,10), $ws_{10.3}$(8,9)
T_9	$ws_{2.3}$(9,11), $ws_{7.3}$(9,10)
T_{10}	$ws_{3.3}$(10,10), $ws_{9.2}$(10,14)
T_{11}	$ws_{4.3}$(11,12)
T_{12}	$ws_{3.4}$(12,14), $ws_{7.4}$(12,13)
T_{13}	$ws_{10.4}$(13,15)
T_{14}	$ws_{2.4}$(14,14), $ws_{8.4}$(14,16)
T_{15}	$ws_{1.4}$(15,19), $ws_{4.4}$(15,16), $ws_{5.4}$(15,18), $ws_{7.5}$(15,17)
T_{16}	
T_{17}	$ws_{2.5}$(17,18), $ws_{3.5}$(17,19)
T_{18}	$ws_{6.4}$(18,20)
T_{19}	$ws_{4.5}$(19,19), ws$_{7.6}$(19,19), $ws_{8.5}$(19,20), $ws_{10.5}$(19,20)
T_{20}	$ws_{9.3}$(20,20)

TABLE 3
UPDATED MINI WEB SERVICES

Start task	Mini-web services	Start task	Mini-web services
T_1	$ws_{4.1}$(1,3)	T_{11}	$ws_{4.3}$(11,12)
T_2	$ws_{5.1}$(2,4)	T_{12}	$ws_{3.4}$(12,14)
T_3	$ws_{3.1}$(3,5)	T_{13}	$ws_{10.4}$(13,15)
T_4	$ws_{6.2}$(4,6)	T_{14}	$ws_{8.4}$(14,16)
T_5	$ws_{4.2}$(5,7)	T_{15}	$ws_{1.4}$(15,19)
T_6	$ws_{2.2}$(6,7)	T_{16}	
T_7	$ws_{3.2}$(7,8)	T_{17}	$ws_{3.5}$(17,19)
T_8	$ws_{6.3}$(8,13)	T_{18}	$ws_{6.4}$(18,20)
T_9	$ws_{2.3}$(9,11)	T_{19}	$ws_{8.5}$(19,20)
T_{10}	$ws_{9.2}$(10,14)	T_{20}	$ws_{9.3}$(20,20)

The number of Web service hop count is six, as shown in Figure 5 and this is one optimal result.

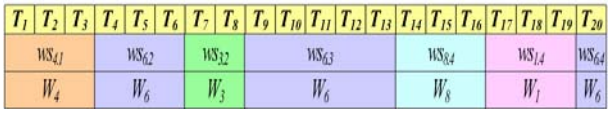

Figure 5 Tasks and their Web service assignments

5. Conclusion & Future work

Due to the popularity of Web services and that the tendency for application functionality is getting more

and more complicated, we propose a problem called LOWSIP which seeks to minimize the hop count between Web services while integrating Web services to serve a set of linearly ordered tasks. A polynomial-time Web service integration algorithm which optimally assigns the Web services to tasks waiting to be executed is presented. This algorithm can find a way to integrate Web services so that the hop count between Web services is minimized. This way, we can speed up the processing time of these tasks.

In the future, we will investigate the effectiveness of different Web services integration methods on other non-functional factors in order to research a more efficient and comprehensive way of integrating the Web Services.

6. References

[1] Munindar P. Singh and Michael N. Huhns, "Service-Oriented Computing: Semantics, Processes, Agents" John Wiley & Sons, Ltd., 2005

[2] Maurizio, A. Sager, J. Corbitt, G. Girolami, L. "Service Oriented Architecture: Challenges for Business and Academia", in Proc. The 41st Annual Hawaii international Conference on System Sciences, 2008. Jan. 2008, pp. 315-322

[3] MSDN Architecture Center," Service Oriented Architecture" Available: http://msdn.microsoft.com/en-us/architecture/aa948857.aspx

[4] Sun, "Service-Oriented Architecture (SOA)". Available: http://www.sun.com/products/soa/index.jsp

[5] IBM," Service Oriented Architecture — SOA". Available: http://www-306.ibm.com/software/solutions/soa/

[6] Oracle,"Oracle Service-Oriented Architecture". Available: http://www.oracle.com/technologies/soa/index.html

[7] Arntzen, Aurelie Aurilla Bechina, "Services Oriented Architecture: integration requirements". In Proc. The 31st Annual international Computer Software and Applications Conference, 2007. COMPSAC, 2007. July 2007. pp.619 - 620

[8] W3C. "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)", available: http://www.w3.org/TR/soap12-part1/

[9] W3C, " Web Services Description Language (WSDL)", Version 2.0, 2007. available: http://www.w3.org/TR/wsdl20/

[10] OASIS, "UDDI Spec TC". Available: http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

[11] Heqing Guan, Beihong Jin, Jun Wei, Wei Xu, Ningjiang Chen, "A framework for application server based Web services management". In Proc. The 12th Asia-Pacific Software Engineering Conference, 2005. APSEC, 2005. Dec. 2005 Page(s):8

[12] Fu-Chiung Cheng, Tai-Chang Hung, Young-Jang Chiou, Te-Chun Chang, " Design and implementation of Web service integration tool" in *Proc. IEEE* International Workshop on Service-Oriented System Engineering,2005. SOSE 2005. Oct. 2005, pp. 91–96

[13] Mohanty, Hrushikesha, Mulchandani, Jitesh, Chenthati, Deepak, Shyamasundar, R.K. "Modeling Web Services with FSM Modules". In Proc. First Asia international Conference on Modelling & Simulation, 2007. AMS, 2007. March 2007 Page(s):100 - 105

[14] Kezhe Tang, Shiping Chen, Levy, D. John Zic, Zic, J. "A Performance Evaluation of Web Services Security". In Proc. The 10th IEEE international Enterprise Distributed Object Computing Conference, 2006. EDOC, 2006. Oct. 2006 Page(s):67 - 74

[15] Sidharth, Navya, Liu, Jigang, "A Framework for Enhancing Web Services Security". In Proc. The 31st Annual international Computer Software and Applications Conference, 2007. COMPSAC, 2007. July 2007. Page(s):23 - 30

[16] Gutierrez, C. Fernandez-Medina, E. Piattini, M. "PWSSec: Process for Web Services Security". In Proc. International Conference on Web Services, 2006. ICWS, 2006. Sept. 2006 Page(s):213 – 222

[17] Zhengping Wu, Weaver, A.C. "Using Web Services to Exchange Security Tokens for Federated Trust Management". In Proc. IEEE International Conference on Web Services, 2007. ICWS, 2007. July 2007 Page(s):1176 -1178

[18] Jian-Xin Li, Bin Li, Liang Li, Tong-Sheng Che, "A Policy Language for Adaptive Web Services Security Framework" in Proc.Eighth ACIS internationalConference on Software Engineering, Artificial Inteligence, Networking, and Parallel/Distributed Computing, 2007. July 30 2007-Aug. 1 2007 Page(s):261 - 266

[19] Abramowicz, W. Kaczmarek, M. Zyskowski, D. "Duality in Web Services Reliability". In Proc. International Conference on Internet and Web Applications and Services/Advanced International Conference on Telecommunications, 2006. AICT-ICIW, 2006. Feb. 2006 Page(s):165 – 165

[20] Hangjung Zo, Nazareth, D.L. Jain, H.K." Measuring Reliability of Applications Composed of Web Services". In Proc. The 40Th Annual Hawaii International Conference on System Sciences, 2007. HICSS, 2007. Jan. 2007 Page(s):278c - 278c

[21] Chan, P.W. Lyu, M.R. Malek, M. "Reliable Web Services: Methodology, Experiment and Modeling". In Proc. IEEE International Conference on Web Services, 2007. ICWS, 2007. July 2007 Page(s):679 – 686

[22] M. R. Garey and D. S. Johnson. "Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)" . W. H. Freeman (January 15, 1979)

[23] H. Agrawal, "Dominators, super blocks, and program coverage," *21st ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, Portland, Oregon, 1994

[24] H. Agrawal, "Efficient Coverage Testing Using Global Dominator Graphs," *1999 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, Toulouse, France, 1999.

[25] V. Chvatal. "A Greedy Heuristic for the Set-Covering Problem." *Mathematics of Operations Research*. 4(3), August, 1979.

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, "Introduction to Algorithms", *MIT Press,* Second Edition, September 2001.

[27] M.J. Harrold, R. Gupta and M.L. Soffa, "A Methodology for Controlling the Size of a Test Suite," *ACM Transactions on Software Engineering and Methodology*, 2(3):270-285, July, 1993.

[28] M. Marre and A. Bertolino, "Using Spanning Sets for Coverage Testing," *IEEE Transactions on Software Engineering*, 29(11):974-984, Nov. 2003.

[29] Sriraman Tallam and Neelam Gupta, "A Concept Analysis Inspired Greedy Algorithm for Test Suite Minimization", *ACM* SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE 2005), Lisbon, Portugal, September 5-6, 2005.