ISSN: 2321-7782 (Online)
Volume 3, Issue 2, February 2015
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com

Replanning Technique of web Service Composition

Savan M. Patel
M.Tech. (CE) Dept.
U.V.Patel Engg. College
Kherva, India

Abstract: Web Services have become a standard for integration of systems in distributed environment.The Service-Oriented Computing (SOC) paradigm refers to the set of concepts, principles, and methods that represent computing in Service-Oriented Architecture (SOA) in which software applications are constructed based on independent component services with standard interfaces. One of the most important benefits of Service-Oriented Computing is to foster the satisfaction of end-user needs through the automatic generation of composite services out of simpler services existing in the user environment. Different approaches have been proposed in the last years to address this issue, e.g., based on model-checking or AI planning. Still, these approaches do not cope with the inherent dynamicity of the service pervasive world, where not only available services, but also user needs, may evolve over time. Setting up service composition in an AI planning framework, we used replanning techniques enabling service compositions to adapt at run-time, both to service and requirement changes, paving the way for on-demand and sustainable end-user service composition. We compare replanning technique with the web service composition algorithm.

Keywords: Service composition, Service-Oriented Computing, Web Services, Semantic Web Services. Replanning technique.

I. Introduction

Web services, adopted by Service Oriented Architecture (SOA), are loosely coupled reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over the internet. It should facilitate integration of newly built applications both within and across organizational boundaries avoiding difficulties due to different platform, heterogeneous programming language. Web services can be used alone or in conjunction with other web services to carry out a complex aggregation or a business transaction. A web service is described using a standard that provides all of the details necessary to interact with the service such as, message formats, transport protocols, and location.

1. *Web Services*- Web services defined by software system to support machine-to-machine interaction over a network. The W3C has a more elaborate definition:[3]

   "A Web service is a software system designed to support interoperable machine to machine interaction over a network. It has an interface described in a machine process able Format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards."

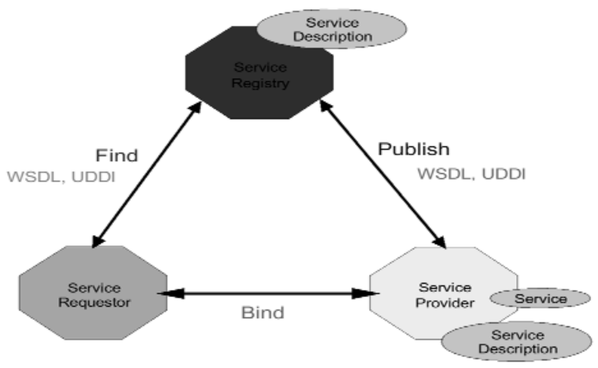


Fig.1 Web Service Model[6].

2. *Service providers*- They use a traditional a programming language such as Java, C++, or C# to write program components. All components will be wrapped with open standard interfaces, call services, or Web services if the services are available over the internet, so that application builders can simply use the services without further communication with the service providers. The same services can be used by many applications.

3. *Service brokers*- Allow services to be registered and published for public access. Help application builders to find services they need.

4. *Application builders*- Instead of constructing software from scratch using basic programming language constructs, the application builders represent the end users to specify the application logic in a high-level specification language, using standard services as components. Application builders are software engineers who have a good understand of software architecture and the application domain.

5. *Web Services Composition*- Individual web services cannot satisfy all the service requests. It becomes necessary to combine functionality of several web services to full fill the need of a given client or when the implementation of a web service's business login involves the invocation of other web services. Such a service built from multiple web service is called a composite service and the process of developing a composite service is called service composition. The components of a composite service can in turn be an elementary service or a composite service.[31]

II. Background and Related Work

We have used replanning algorithm focused at obtaining an updated service composition solution efficient rather than obtain in gall possible modified service composition solutions. We will use two Qos parameters namely Response time and Throughput. We will use these Qos parameters to compare our replanning algorithm with Qos parameters and without Qos parameters. Replanning which is the standard planning technique for plan modification. We will compare our replanning algorithm with the standard greedy algorithm.

A. *Web Service Composition Algorithm*

Algorithm uses heuristics to search for a repaired composition. The algorithm starts from the goal level. It tries to satisfy the unimplemented goals first (Line 2 to 8). When new services are added into the partial planning graph, its precondition may not be satisfied. The unsatisfied preconditions are added to U to be satisfied at a lower level (Line 8, 17 and 29). This process goes from the goal level toward the initial proposition level (Line 10 to 18). It is possible that after adding actions at A1, we still have some broken preconditions. In this case, we need to add new levels (Line 19 to 32). We use UH to record the history of preconditions we have satisfied. If $UH \cap U \neq ø$, that means some precondition broken for the second time. It is a deadlock situation. We stop the search. This Algorithm is a best first search algorithm. It does not generate a full planning graph, but rather, to fast fix the broken graph and obtain a feasible solution. It is possible that algorithm does not find a solution to a problem with solutions. It is possible that algorithm generates a graph that contains multiple solutions. Therefore, a backwardsearch() function returns the first obtained solution. We know that a solution can be found from the graph made of the originally broken plan and the newly added services. backwardsearch() is on this very small graph, thus fast.

III. Problem Formulation and Algorithm

Following are some definitions and equations that are define in our approach for formulating the replanning technique.

1). *Definition 1 (web service (w)): "A Web service is a software system designed to support interoperable machine to machine interaction over a network."* A web service whose inputs are **in(w)** and whose outputs are **out(w).**

2). *Definition 2 (Partial Planning Graph (G)):* We first identify the new composition problem with the new set of available services and new goals. The disappeared services are removed from the original planning graph and new goals are added to the goal level, yielding a partial planning graph. The repair algorithm "regrows" this partial planning graph wherever the heuristic function tells the unimplemented goals and broken preconditions can be satisfied.

3). *Definition 3 (Unimplemented goals (g)): **g** is the set of* unimplemented goals.

4). *Definition 4 (Unsatisfied preconditions (U)):* **U** is a set of unsatisfied preconditions (inputs) of some services in *Partial Planning Graph (G).*

5). *Definition 5 (Initial level and Final level (Lin and Lout)):* Lin is the initial proposal level noted as P0; the first action level is noted as A1; the last proposition level Pn which is also the Lout level.

6). *Equation 1:* Assume we want to add an action w to the highest action level n, the evaluation function is:

$$\mathbf{f(G,w)=|g \cap out(w)| * 10 + |Pn\text{-}1 \cap in(w)| - |in(w) - Pn\text{-}1| - e(w,G)}$$

Equation 1[12]

Where $|\breve{g} \cap out(w)|$ is the number of unimplemented goals that can be implemented by w. The coefficient 10 is the weight of this term. It shows that to satisfy the goals is more important than the other needs represented by the following terms; $|Pn\text{-}1 \cap in(w)|$ is the number of the inputs of w that can be provided by the known parameters at the level Pn-1; $|in(w) - Pn\text{-}1|$ is the number of the inputs of w that cannot be provided by the known parameters at the level Pn-1. This set needs to be added into U, if w is added. e(w,G) is the number of the actions in G that are exclusive with w.

7). *Equation 2:* Assume we want to add an action w to action level m, and m is not the goal level, the evaluation function is:

$$\mathbf{f(G,w)=|g \cap out(w)| * 10 + |Pm\text{-}1 \cap U \cap out(w)| + |Pm\text{-}1 \cap in(w)| - |in(w) - Pm\text{-}1| - e(w,G)}$$

Equation 2[12]

Compared to equation 1, the above equation added term $|Pm \cap U \cap out(w)|$ which is the number of the broken propositions in level Pm that can be satisfied by the outputs of w.

Qos Parameters: There are following Qos parameters used in algorithm.

1). *Throughput:* It is critical for service consumers to know the amount of work that a service can perform in a given period of time (e.g., number of requests per second). For example, in airline booking services, intensive inquiries are often inputted within a short period of time, so it is important for consumers of such service to ensure that service's throughput can fulfill an anticipated volume of requests. Throughput of a service, S, can be represented as follows.

Throughput (S) = Number of requests / per unit-of-time

According to the service's granularity, the unit-of-time may vary from mini-second to minute. As well as response time, a flexible description method is required to adapt throughput descriptions to different services.

2). *Response time:* Response time is a typical measurable performance attribute that refers to the elapsed time between the initiation of a service request and the completion of the service's response. The evaluation of response time usually consists of execution time and waiting time. A service's response time for a request, *R*, can be represented as shown below.

Response time(R) = Execution time(R) + Waiting time(R)

The execution time is the duration of performing service functionality. The waiting time is the amount of time for all possible mediate events such as message transmissions between service consumers and providers. However, the evaluation of response time is controversial due to the uncertainty of network fluctuations. From service consumer perspective, it is meaningful to consider response time as the duration starting from the issue of a request to the end of receipt of a service's response. But from service provider perspective, response time is considered as same as execution time of a service, so it does not include all possible mediate events, which are seen as incontrollable variables during service execution. The gap is because of the fact that service providers cannot precisely describe the waiting time of a service execution. In order to minimize the gap, a flexible description method is required to balance the two viewpoints.

A). Replanning Algorithm

Input: W, G, U, g defined as before

Output: either a plan or fail

```
1:  result = fail then stop and return fail otherwise going to next step
2:  while any unimplemented goal (g) is remaining in set of g then do
3:      select web service (w) with the f(G,w) according to equation 1
4:      if w does not exist then break
5:      add w to G at action level An
6:      add out(w) at proposition level Pn
7:      remove satisfied goal from g
8:      add in(w) – Pn-1 to U( unsatisfied preconditions)
9:  if g ≠ø then return result
10: for m = n -1; m> 0; m -- do
11:     while U ∩ Pm ≠ø  do
12:             select web service (w) with the f(G,w) according to equation 2
13:             if w does not exist then break
14:             add w to G at action level Am
15:             add out(w) at proposition level Pm
16:             remove satisfied preconditions from U
17:             add in(w) – Pm-1 to U
18:     if U ∩ Pm ≠ø then return result
19: UH = U
20: while any unsatisfied preconditions remaining(U) and any web service is remaining then do
21:     insert an empty proposition level (PE) and empty action level A1
22:     PE = P0 - U
23:     while U ∩ PE ≠ø do
24:             select an action w with the best f(G,w) according to equation 2
25:             if w does not exist then break
26:             add w to G at action level A1
27:             add out(w) at proposition level PE
28:             remove satisfied preconditions from U
29:             add in(w) - P0  to U
30:             remove w from W
31:     if U ∩ PE ≠ø; then break
32: if U ∩ UH ≠ø then break else add U to UH
33: if U ≠ø then return result
34: result = backwardsearch(G)
35: return result
```

Algorithm Description

- Algorithm uses heuristics to search for a repaired composition.
- The algorithm starts from the goal level. It tries to satisfy the unimplemented goals first (Line 2 to 8).
- When new services are added into the partial planning graph, its precondition may not be satisfied.
- The unsatisfied preconditions are added to U to be satisfied at a lower level (Line 8, 17 and 29).
- This process goes from the goal level toward the initial proposition level (Line 10 to 18).
- It is possible that after adding actions at A1, we still have some broken preconditions. In this case, we need to add new levels (Line 19 to 32).

- We use UH to record the history of preconditions we have satisfied. If UH ∩ U ≠ ø, that means some precondition broken for the second time. It is a deadlock situation. We stop the search.

This Algorithm is a best first search algorithm. It does not generate a full planning graph, but rather, to fast fix the broken graph and obtain a feasible solution. It is possible that algorithm does not find a solution to a problem with solutions. It is possible that algorithm generates a graph that contains multiple solutions. Therefore, a backwardsearch() function returns the first obtained solution. We know that a solution can be found from the graph made of the originally broken plan and the newly added services. backwardsearch() is on this very small graph, thus fast.

B). Flowchart

Web service composition replanning Algorithm depend on No. of service in repository so as we increases no. of service execution time is also increase. The time complexity in the worst case is O(b^{m}). Where b is the maximum number successors of any nodes, namely, the number of services that can produce any concept required by the candidate composition, m is the maximum depth of the search space, namely, the maximum number of services in a composition.

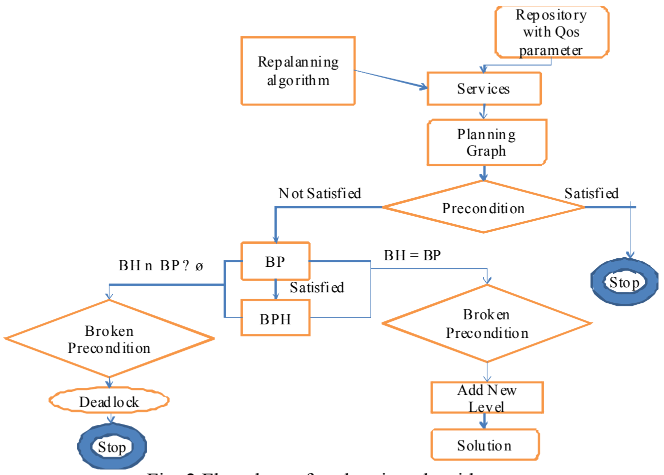


Fig. 2 Flowchart of replanning algorithm

IV. Experimental Setup

A). Web Service Challenge (WSC) Test Set

- The web service challenge started in 2005 to stimulate research into web service composition, growing and evolving each year.
- During the years 2005 to 2007, the Web Service Challenge focused on optimizing the service discovery and composition process solely, using abstractions from real-world situations. The taxonomies of semantic concepts as well as the involved data formats were purely artificial.
- Starting with the 2008 competition, the data formats and the contest data itself will be based on the OWL, WSDL, and WSBPEL schemas for ontologies, services, and service orchestrations to mimic real world scenario.
- In 2009, each service is annotated with nonfunctional properties. The Quality of Service (QoS) of a Web Service is expressed by values expressing its response time and throughput.

B). Hardware Configuration:

All experiments were performed on a PC platform with a Intel(R) Core(TM) i3 CPU (2.40 GHz), Windows 7 Home Basic, and 3.00 GB RAM, 64 bit Opearating System All algorithms were implemented in Java with the use of the Eclipse tool.

C). Experiment Results With Comparison:

a). Comparison of without Qos and with Qos replanning algorithm results

Dataset No.	No. of Service in dataset	No. of concepts in dataset	No service in Compsition	Execution Time(Mill.sec)
1	572	1578	3	15
2	4129	12388	6	16
3	8138	18573	3	31
4	8301	18673	5	62

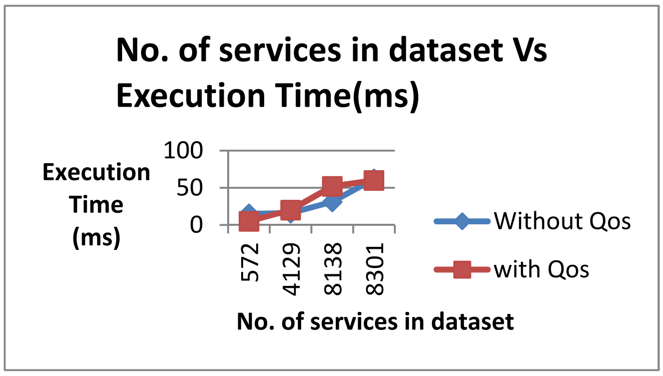
Table 1.1- without Qos replanning algorithm result

Dataset No.	No. of Service in dataset	No. of concepts in dataset	No service in Composition	Execution Time(ms)	Throughtput		Response time	
					min	Max	min	max
1	572	1578	3	5	1000	20000	10	500
2	4129	12388	6	20	1000	20000	10	500
3	8138	18573	3	52	1000	20000	10	500
4	8301	18673	5	60	1000	20000	10	500

Table 1.2- without Qos replanning algorithm results

Analysis

The above comparison of performance is recorded in the cases that both with Qos and without Qos parameters replanning algorithms can find solutions.Figure show the results from Experiment 1. From Figure we can see that the without replanning algorithm execution time is slightly decreasing when more Web services are removed. It is because the with Qos replanning service composition algorithm check the Qos parameter from the services so it takes some time. Minimum time for the throughput Qos parameter is 1000 millisecond and maximum time is 20000 millisecond. Minimum time for the response time Qos parameter is 10 millisecond and maximum time is 500 millisecond.



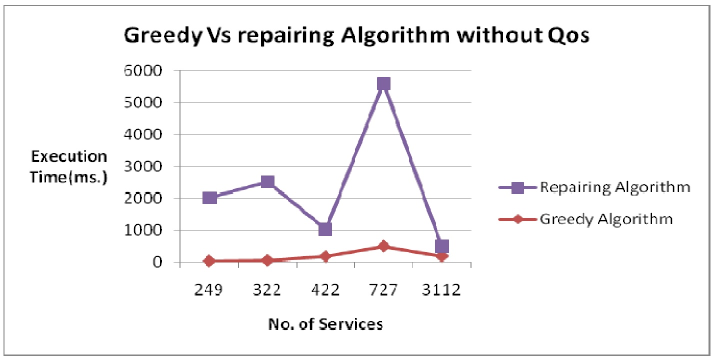
b). Comparison of Greedy Algorithm with Repairing Algorithm

Data Set	No. Service	No. Concept	No.Service in Com-position	Execution time
				Without Qos
1	249	173	10	23
2	322	239	10	44
3	422	273	10	167
4	727	366	10	494
5	3112	1223	10	175

Table 2.1 Greedy algorithm results

Data Set	No. Service	No. Concept	No. Service in Com-position	Execution time
				Without Qos
1	249	173	10	1985
2	322	239	10	2460
3	422	273	10	853
4	727	366	10	5113
5	3112	1223	10	311

Table 2.2 Replanning algorithm results



Analysis

Figure show the number of services and execution time for composition in a composed plan. Our explanation is that our repair algorithm does not work faster than the greedy algorithm because replanning algorithm takes time to finds a solution; the quality of the solution is pretty good. When services are 727 then require execution time is exponentially increased and when services are 3112 then it exponentially decrease.

V. Conclusion and Future Work

Service compositions have to be adapted whenever the composition context, *i.e.,* available services and composition requirements, change. We have set up service composition in the AI planning domain and we have used a planning graph repair algorithm focused at obtaining an updated service composition solution fast rather than obtaining all possible modified service composition solutions. We observed that when services are removed on a percentage basis of the whole service set, and when available services are more and similar the repaired plan can be as good as the one obtained with replanning but is more similar to the original plan.

To improve the repair algorithm following existing work in replanning that propose, upon disappearance of some service(s)**.** To remove even more services in order to jump out of local extrema in the search for a new plan.

References

1. W3C. SOAP 1.2 Working draft, 2001. http://www.w3.org/TR/2001/WD-soap12-part0-20011217
2. UDDI Consortium. UDDI Executive White Paper, Nov. 2001. http://uddi.org/pubs/UDDI_Executive_White_Paper.pdf
3. T. Berners-Lee, J. Hendler, and O. Lassila, The Semantic Web, Scientific American May 2001.
4. E.Christensen, F.Curbera, G.Meredith and S.Weerawarana., Web Services Description Language (WSDL) 1.1, 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315
5. S.McIlraith, T.C.Son and H.Zeng., "Semantic Web Services", IEEE Intelligent Systems, 16(2), Mar.2002.
6. Antonio Bucchiarone 1, Stefania Gnesi, "A survey on Service Composition Languages and Model", International Workshop on Web Services Modeling and Testing (WS-MaTe 2006).
7. G.Alonso, F.Casati, H.Kuno and V.Machiraju., "Web Services: Concepts, Architectures and Application", Springer-Verlag Berlin Heidelberg 2004.
8. M.Dean, D.Connolly, F.van Harmelen, J.Hendler, I.Horrocks, D.L.McGuinness, P.F.Patel-Schneider and L.AStein., "Web Ontology Language (OWL) " Reference Version1.0, W3C Working Draft 12 November 2002.
9. Schahram Dustdar, Wolfgang Schreiner, "A survey on web services composition", International Journal of Web and Grid Services, Vol. 1, No. 1, 2005.
10. Jorge Cardoso, "Quality of Service and Semantic Composition of Workflows", PhD thesis, Department of Computer Science, University of Georgia, Athens, GA (USA), 2002.
11. Zeng Liang zhao, Benatallah B, Dumasm, et al., "Quality driven Web services composition", in Proc. of the12th International Conference on World Wide Web. New York: ACM Press, 2003: 411-421.
12. Yuhong Yan, Pascal Poizatyz and Ludeng Zhao., "Repairing Service Compositions in a Changing World", Concordia University, Montreal, Canada, zLRI UMR 8623 CNRS, Orsay, France.
13. Weigand, Willem-Jan van den Heuvel and Marcel Hiel, "Rule-Based Service Composition and Service-Oriented Business Rule Management", INFOLAB, Tilburg University, Warandelaan 2, Tilburg, The Netherlands.
14. T. Weise, S. Bleul, D. Comes, and K. Geihs. "Different approaches to semantic web service composition", In ICIW '08: Proc. of the 2008 3rd International Conference on Internet and Web Applications and Services, pages 90–96, Washington, DC, USA, 2008. IEEE Computer Society.
15. Frederico G. Alvares de Oliveira Jr., Jos´e M. Parente de Oliveira, "QoS-based Approach for Dynamic Web Services Composition", Journal of Universal Computer Science, vol. 17, no. 5 (2011), 712-741.
16. S. Kambhampati, E. Parker, and E. Lambrecht. Understanding and Extending Graphplan. In S. Steel and R. Alami, editors, Proc. of ECP, volume 1348 of Lecture Notes in Computer Science, pages 260–272.Springer, 1997.
17. S.-C. Oh, D. Lee, and S. Kumara. Web Service Planner (WSPR): An Effective and Scalable Web Service Composition Algorithm. Int. J. Web Service Res., 4(1):1–22, 2007.
18. S. Chandrasekaran, S. Madden, and M. Ionescu, "Ninja workflows: An architecture for composing services over wide area networks," Univ. California, Berkeley, CA, CS262 class project writeup, 2000.
19. S. Ghandeharizadeh ,Craig A. Knoblock , C. Papadopoulos , E. Alwagait , C. Shahabi, "Proteus: A System for Dynamically Composing and Intelligently Executing Web Services", in Proc. 1stInternational Conference on Web Services (ICWS), Las Vegas, Nevada, 2003.
20. Debra Vandermeer, "FUSION: a system allowing dynamic web service composition and automatic execution", in Proc. of IEEE International Conference on E-Commerce (CEC'03) Athens, Greece, 2003.
21. Shankar R. Ponnekanti and Armando Fox, "SWORD: A developer toolkit for web service composition", in Proc. 11th World Wide Web Conference, Honolulu, Hawaii, May 7-11, 2002.
22. E. Sirin and B. Parsia, "Planning for semantic web services," in Proc. Semantic Web Services Workshop 3rd Int. Semantic Web Conf., 2004.
23. M. Minami, H. Morikawa, and T. Aoyama, "The design and evaluation of an interface-based naming system for supporting service synthesis in ubiquitous computing environment," Trans. Inst. Electron., Inf.Commun. Eng., vol. J86-B, no. 5, pp. 777–789, May 2003.
24. Q. Z. Sheng, B. Benatallah, M. Dumas, and E. Mak, "SELF-SERV: A platform for rapid composition of web services in a peer-to-peer environment," in Proc. 28th Very Large Database Conf., Hong Kong, China, Aug. 2002.
25. Hongbing Wang,Joshua Zhexue Huang, Yuzhong Qu, Junyuan Xie , "Web services: problems and future directions", Journal of Science Direct.

26. F. Lautenbacher, B. Bauer, "A Survey on Workflow Annotation and Composition Approaches", In M. Hepp, K. Hinkelmann, D. Karagiannis, R. Klein, N. Stojanovic (eds.) Semantic Business Process and Product Lifecycle Management, in Proc. of the Workshop SBPM 2007, Innsbruck, April 7, 2007, CEUR Workshop Proceedings, ISSN 1613-0073, online CEUR-WS.org/Vol-251.

27. Kaouthar Boumhamdi, Zahi Jarir , "A Flexible Approach to Compose Web Services in Dynamic Environment", International Journal of Digital Society (IJDS), Volume 1, Issue 2, June 2010.

28. Sleiman Rabah, Dan Ni, Payam Jahanshahi, Luis Felipe Guzman (2011), "Current State and Challenges of Automatic Planning in Web Service Composition", Department of Computer Science and Software Engineering Concordia University Montréal, Québec, Canada.

29. David Martin , Mark Burstein , Drew McDermott ,Sheila McIlraith , Massimo Paolucci Katia Sycara , Deborah L. McGuinness , Evren Sirin , Naveen Srinivasan, "Bringing Semantics to Web Services with OWL-S", Springer Science July 2007.

30. George Baryannis and Dimitris Plexousakis, "Automated Web Service Composition: State of the Art and Research Challenges", Technical Report ICS FORTH/TR-409 October 2010.

AUTHOR(S) PROFILE

Savan Patel, received the Bachelor of Engineering degree in Computer Engineering from Sankalchand Patel College of Engineering, Visnagar under Gujarat Technological University Ahmedabad in 2012 and Completed Master of Technology in Computer Engineering from U.V.Patel College of Engineering, Mehsana.